An Architecture for Multilevel Learning and Robotic Control based on Concept Generation

Thesis

An Architecture for Multilevel Learning and Robotic Control based on Concept Generation

by Pejman Iravani

Submitted to the Department of Computing
in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

at The Open University

March 2005

An Architecture for Multilevel Learning and Robotic Control based on Concept Generation

by

Pejman Iravani

Abstract

Robot and multi-robot systems are inherently complex systems, for which designing the programs to control their behaviours proves complicated. Moreover, control programs that have been successfully designed for a particular environment and task can become useless if either of these change. It is for this reason that this thesis investigates the use of machine learning within robot and multi-robot systems. It explores an architecture for machine learning, applied to autonomous mobile robots based on dividing the learning task into two individual but interleaved sub-tasks.

The first sub-task consists of finding an appropriate representation on which to base behaviour learning. The thesis explores the viability of using multidimensional classification techniques to generalise the original sensor and motor representations into abstract hierarchies of 'concepts'. To construct concepts the research used standard classification techniques, and experimented with a novel method of multidimensional data classification based on 'Q-analysis'. Results suggest that this may be a powerful new approach to concept learning.

The second sub-task consists of using the previously acquired concepts as the representation for behaviour learning. The thesis explores whether it is possible to learn robotic behaviours represented using concepts. Results show that is possible to learn low-level behaviours such as navigation and higher-level ones such as ball passing in robot football.

The thesis concludes that the proposed architecture is viable for robotic behaviour learning and control, and that incorporating Q-analysis based classification results in a promising new approach to the control of robot and multi-robot systems.

ALL MISSING PAGES ARE BLANK

IN

ORIGINAL

Contents

List of Figures

List of Tables

Acknowledgements

Although PhD research is considered mostly to be the work of an individual, this often happens in the context of a *team*. Part of my team were my supervisors, Jeffrey Johnson and Lucia Rapanotti, who contributed to this thesis providing me guidance in the research process and many technical comments related to this thesis. I thank them for their help and continuous encouragement.

I would also like to thank Marian Petre for organising and running the postgraduate research forums at the Computing Department. These forums helped me understanding what a PhD is about. Also to Tony Hirst who proofread this thesis and provided many useful comments.

More personally, during these last four years I've shared many experiences with many good friends. Firstly, I would like to thank the people who shared the Atisha house, specially to Juanjo, Manoj, Tawanda, Narcís, Quentin, Miquel, Enrico, Tim, Jirka, Niall and Luis for all dinners, parties, political discussions and midnight pasta shared during this time.

I would also like to thank all the students at the Open University, specially Jose Luis, Manuel, Joan, Iestin, Sumit, Avik, Katerina, and Jonathan who in one way or another always had a word of advise and encouragement.

Finally, I thank Aurelíe, my parents Rosa and Bahman, my sister Jasmin and my brother Shayan for always being the strongest support I always had and still have.

Pejman Iravani

March 24, 2005

Chapter 1

Introduction

This thesis is concerned with the design of autonomous robotic systems. Robotics systems are complex, in the sense that many loosely coupled parts interact to produce the system's final behaviour. For example, in mobile robots the effect of motors, sensors, noise, friction, inertia, mechanical components, changing environments, etc. play a role in the current and future state of the system. In such a context, and because the system can not be easily modularised, modelling, prediction and control of such systems prove to be *complex.*

This complexity manifests itself in various manners. For example, in the *combinatorial nature* of the control of robot systems which, like in chess, it proves impossible to plan moves into the distant future. Moreover, robot systems are also *chaotic* in the technical sense: start a robot from exactly the same state and give it exactly the same command, and its trajectories will deviate from previous observations.

This complexity means that it is impossible for the designer to foresee all the possible effects of the interactions between the robot, the task and the

environment. This poses serious difficulties in trying to program robots *a priori* and means that robots must *learn* from experience.

1.1 Motivation

We are interested in designing robotic agents which are capable of solving tasks and achieving goals autonomously, where autonomy is the capability of acting independently, as further explained in Section 2.1.1.

Although there is no single accepted definition in the literature of what an agent is, here it is defined as a physical or simulated entity which is capable of acting in an environment autonomously. A more detailed definition of agent is also given in Section 2.1.1. In this thesis we sometimes use the term agent to mean a robot, when we want to emphasise its more abstract properties.

Many issues arise in the design of fully autonomous agents. This thesis focuses on one of them, namely *adaptation*, which is the capability by which an agent can transform its 'way of acting' according to changes in its environment. For example, if a mobile robot is encountering problems in a navigation task (e.g. bumping into obstacles), then if by changing some of its properties (e.g. motion speed) it achieves a desired behaviour, we say that the agent has adapted to the environment. An agent with adaptive capabilities will be easier to design, as unforseen situations can be dealt by the agent itself.

As will be discussed later, adaption capabilities can be achieved by having agents that *learn* from experience. Learning is a mechanism by which the agent adapts to undesired characteristics, such as the one mentioned previously.

Finally, the main focus in this thesis is learning in artificial robotic systems, and as it will be seen in the next chapter, for learning methods to be practically applicable they require *generalisation* of experience and 'knowledge'.

1.2 Research question

The main research question addressed in this thesis is:

> Can artificial robotic agents learn generalised entities, known as *concepts*, using raw sensory-motor data and their interaction with the environment? If so, is it possible to integrate such concepts in a multilevel architecture which allows for behaviour learning and robot control?

More precisely, this question can be decomposed into the following parts:

- Question **Q1:** Is it possible to use robotic sensor and motor data to learn abstract entities called *concepts*?

- Question **Q2:** Is it possible to use such *concepts* as the representation for learning robotic behaviours? Do such concepts provide any benefits for behaviour learning? In particular, how do they address the generalisation problems faced in machine learning?

- Question **Q3:** Is it possible to control autonomous mobile robots using this notion of *concept*?

- Question **Q4:** Is it possible to integrate the notion of *concept* within a multilevel architecture which exploits the definition of concepts for learning and control?

1.3 Arguments of the thesis

In order to answer the question in the previous section, this thesis makes the following arguments:

Question **Q1**:

- Argument **A1:** In general, autonomous robots gather information from the environment using their sensors. The information that robotic sensors provide is characterised by being highly dimensional, noisy and only partially observable. For example, a mobile robot could use a wide variety of sensor devices, such as, a sonar, infrared light sensors, cameras, bumpers, encoders, etc. resulting in a high dimensional sensory input. Also, the measurements could be inaccurate as different material surfaces, light conditions, frictions, etc. change the response of sensors. Finally, sensors provide only partial information, i.e. local and incomplete information about the environment.

  It will be demonstrated that, using this type of data and the robot's interaction with the environment, it is possible to acquire general entities or concepts, by applying different classification techniques.

Question **Q2**:

- Argument **A2:** It will be demonstrated that it is possible to develop a learning mechanism that exploits concepts as basic representations for

learning robotic behaviours.

- Argument **A3:** It will be shown that behaviour learning using concepts addresses some of the problems related to continuous and high-dimensional input spaces.

Question **Q3**:

- Argument **A4:** It will be argued that concepts can be used for controlling autonomous robots in a variety of different contexts, including physical or simulated, single or multi-robot.

Question **Q4**:

- Argument **A5:** It will be argued that a multilevel representation is needed for solving complex robot tasks such as in multi-robot soccer, with concepts existing at different levels of description.

- Argument **A6:** It will be demonstrated that it is possible to replicate the multilevel architecture for learning behaviours of different complexity.

The concluding chapter of this thesis explains how these arguments answer the research question.

1.4 Thesis structure

Following is a brief description of the contents in each of the chapters that appear in this thesis.

Chapter 2 introduces the background work related to this thesis, that is: robotic architectures and machine learning approaches.

Chapter 3 defines 'concepts' as classifications of particular multidimensional observations, and reviews some of the existent methods and techniques to classify multidimensional data. It introduces the methodology of Q-analysis that will be used, in a novel manner, to define a new type of concept, known as *relational concept.*

Chapter 4 integrates the work presented in the previous chapters into a multilevel architecture for behaviour learning and robot control based on concept generation.

Chapter 5 experimentally analyses the proposed architecture applied to the learning and control of a robot in a low-level navigation task. In this chapter, the architecture exploits the usage of 'generalisation' concepts.

Chapter 6 gives some experimental results of applying the architecture to a higher-level strategic ball-passing behaviour. In this chapter, the architecture exploits the usage of 'relational' concepts.

Chapter 7 states the conclusions and the contributions of the research. The chapter also discusses a number of open questions raised by the research, suggesting various issues for further research.

Chapter 2

Robotics: Architectures and Learning

This chapter presents a critical review of the literature related to this dissertation, identifying some open questions. Some of these are the focus of our research, and are addressed in the following chapters.

Section 2.1 presents a general introduction to the field of intelligent agents and robots. From this it is identified that adaptability and learning are key characteristics for building flexible and autonomous agents.

Section 2.2 presents the most common robotic architectures, then discusses their limitations, benefits and suitability for supporting adaptability and learning.

Section 2.3 presents the literature related to learning and adaptability in artificial systems. It presents reinforcement learning in greater detail as this is the context for the learning architecture presented in the following chapters.

Section 2.4 discusses issues that complicate the practical application of

machine learning techniques on robotic systems.

Section 2.5 presents some conclusions drawn from the literature review, stating the issues identified for research in this thesis.

2.1 Intelligent agents and robots

This section providing a general definition of autonomous robots or agents, and describes their most important characteristics.

2.1.1 Basic definitions

This thesis defines an *agent* following the definitions in [Jennings *et al.*, 1998, Wooldridge and Jennings, 1995] as an artificial entity, which possesses characteristics such as: *autonomy, social ability, pro-activity* and *real-time actuation.* To this general definition of agent, we add the characteristic of *embodiment* [Chrisley and Ziemke, 2002, Steels, 1995c] to define what we consider to be an autonomous robot. Thus, an *autonomous robot* is an embodied agent.

Before further describing the characteristics that define autonomous robots, it is necessary to remark that, many of the definitions such as autonomy, embodiment, and agency are not 'universal', and different authors define them differently. We have used the descriptions that best suit the purposes of this thesis, which are following:

- **Autonomy** [Steels, 1995b, Steels, 1995a] describe intelligent and autonomous agents from a biological perspective. From this perspective, it is possible to differentiate between automaticity (acting as a stand-

alone system, automatic) and autonomy (capability of forming and adapting its principles of behaviour, autonomous). For example, an automatic system would be one capable of flying a plane, given a prior control strategy (how to fly) and the necessary information to apply the given control (flying path). An autonomous plane would be one capable of changing its prior knowledge in order achieve some internal goals (e.g. changing its flying path or even its flying strategy). As we can see from this example, there are situations in which autonomy needs to be constrained in order to maintain the necessary safety requirements. This thesis supports the view, in which an agent is autonomous if it can adapt its behaviours so that it satisfies some goals. For this definition of autonomy, adaptive behaviour is a key element.

- **Social ability** an agent is said to have social abilities if it is capable of interacting with other agents or humans, so that, the result of this interaction is to maximise collective benefit rather than individual benefit. Social ability has been studied in the context of Multi-Agent Systems (MAS) [Weiss, 1999, Jennings *et al.*, 1998], in which multiple agents must cooperate, coordinate and negotiate in order to achieve and maximise collective benefit. In [Cao *et al.*, 1997] a comprehensive critical review of cooperative mobile robotics is presented, revealing the main issues related to cooperative or team action.

- **Real time actuation** an agent acts in real-time if its actuation achieves control of its behaviour in a timely manner. That is, the agent is capable of reacting to changes in the environment as fast as these happen. For example, a robot with a real-time vision system would be one that

is capable of sensing the movements and changes that occur in its environment.

- **Pro-activeness** an agent is pro-active if it exhibits goal directed behaviour, rather than purely reacting to the stimuli perceived from the environment [Wooldridge and Jennings, 1995]. This characteristic is present in agents that possess deliberative capabilities (i.e. for planning and predicting future situations), and use them to actively select the appropriate actions to take, without need for any environmental stimulus to occur.

- **Embodied** embodiment makes reference to robots which have some hardware implementation (physical bodies) or some software simulations of it, and that are situated in an environment with which they interact. *Embodied robotics*, *embodied artificial intelligence* and *situated cognition* have emphasised that intelligence is not an independent capability of the agent, but that it is related to the interaction between the agent and the environment [Chrisley and Ziemke, 2002, Pfeifer and Scheier, 2001, Steels, 1995c]. This view of intelligence highlights the role that the 'body' and the sensory-motor capabilities of the intelligent agent play in cognition.

2.1.2 Characteristics of robotic systems

The following are some of the characteristics of robotic systems that complicate their control, and which must be taken into account when designing robots. These characteristics are described in any introductory text on robotics, such as [Nehmzow, 2003].

- **Sensor and motor noise** sensors and motors are subjected to certain amount of inaccuracy or error in their readings or actuation due to noise. Simulations of physical robots and environments usually add random noise to make simulations more realistic.

- **Stochastic environments** environments are stochastic. Unlike in the game of Chess, where every action changes the board configuration in a deterministic manner, the effects of a robot's actions are stochastic and depend on the robot-environment interaction.

- **Dynamic environments** environments have dynamic properties, i.e., even if the agent does not perform any action, the environment may change. Changes can be caused by other agents sharing the same environment. For example, in a soccer game, if a player does not act, the game keeps changing as other players keep acting.

- **Partial observability** robots can only perceive the environment partially, there are areas in the environment which their sensors can not access, usually due to sensor-range constrains. For example, a robot soccer player may not perceive where the ball is, as it may be obscured by another player in the field.

These characteristics complicate robotic control, and as will be shown in the next section, robotic architectures need to take them into account in order to produce their control regimes.

2.2 Robot architectures

The selection of an appropriate architecture is an important factor in the design of robotic systems as it provides the system's *structure* and *style* [Coste-Manière and Simmons, 2000]. Structure refers to how the system can be divided into sub-systems and how these interact. Structure is usually represented by some graphical architectural description, where sub-systems are represented by boxes and arrows between them to indicate their interaction. The style refers to the computational concepts that underlie each sub-system, for example detailing how each sub-system operates. This section provides a review of some architectures used to design robots, focusing on their structure and style characteristics.

Three main architectures have been developed for the design of intelligent robots, namely, *deliberative*, *reactive* or *behaviour-based* and *hybrid*, which are described in sections: 2.2.1 to 2.2.3. These sections present the main characteristics of each architectures, focusing on the following: (i) *data representation*, i.e. the way the architecture represents the information it processes. (ii) *Action selection mechanism*, i.e. the mechanisms used for choosing among the possible actions. (iii) *Real-time actuation*, i.e. whether the architecture is capable of real-time actuation in dynamic environments such as robotic environments. (iv) *Goal-directed behaviour*, i.e. whether the actions selected are purely stimulus-response or selected to achieve a specified goal. (v) *Architectural structure*, i.e., how the architecture is decomposed into sub-systems. (vi) *Usage of the agent-environment interaction*, i.e. whether the interaction with the environment is exploited in the control of the agent. (vii) *Applicability of learning*, i.e. how learning is applicable to

each architecture.

Section 2.2.4 provides a general discussion of the weaknesses and strengths of each of the architectures, based on their characteristics.

2.2.1 Deliberative or symbolic architectures

Deliberative architectures, such as IRMA [Bratman *et al.*, 1988] sometimes known also as symbolic, are based on the classic symbolic AI approach, in which the agent operates sequentially according to three steps, namely *sense, plan* and *act.*

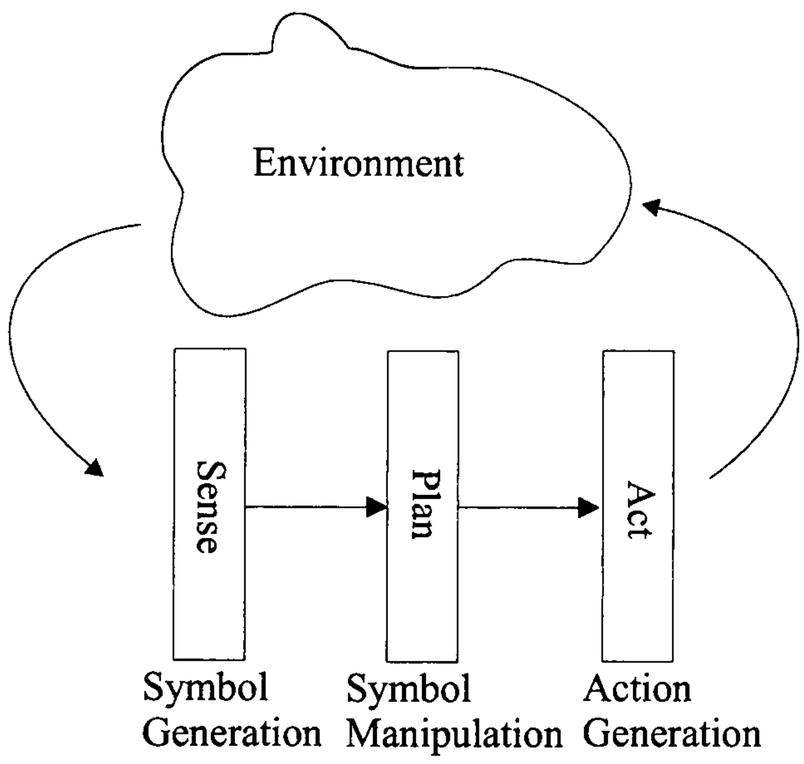


Figure 2-1: General configuration of a deliberative architecture

Figure 2-1 illustrates the general configuration of a deliberative architecture. The three blocks labelled *sense, plan* and *act* are described following.

1. **Sense** the agent observes its environment, and computes a set of symbols and expressions that represent the state of the environment (e.g. a set of objects in the environment and their positions) and some of the agent's internal variables (e.g. the task to carry out).

2. **Plan** the agent uses the previous expressions and *a priori* defined models, which provide information of the robot-environment interaction, to compute plans for achieving the task's goal.

3. **Act** the agent executes the plan.

Deliberative approaches are based on the generation and manipulation of symbols and expressions, as captured by the Physical Symbol System Hypothesis [Newell and Simon, 1976]. The hypothesis states:

> *"The Physical Symbol System Hypothesis: A physical symbol system has the necessary and sufficient means for general intelligent action".*

This hypothesis says that 'general intelligent action' can be obtained by using physical symbol systems. These systems comprise collections of three elements, namely symbols, expressions and processes. *Symbols* are representations of physical patterns that obey physical laws and can be engineered e.g. 'some physical objects'. Some of these symbols can be instantiated e.g. 'physical objects observable at a particular point in time'. Instantiated symbols form *expressions* that indicate the relation between these symbols, e.g. 'physical objects being near to each other'. *Processes* are operations that create, modify, reproduce and destroy expressions. Then, a physical symbol system is a machine that creates and modifies expressions through time.

General purpose computers and robots are examples of physical symbol systems. Robots can compute symbolic expressions which represent, for example, the state of the environment they are observing at a point in time.

Then, they can use planners (explained below) to modify these expressions, so that they achieve a desired state known as the *goal state*.

Planning [Georgeff, 1987] is the main mechanism used to reason in the deliberative approaches. Planners like STRIPS [Fikes and Nilson, 1971], take a symbolic description of the world and of the desired goal state, and they possess a set of action descriptions (operators), expressed as pre- and post-conditions, which they use to compute plans, using some kind of heuristics; for example, means-ends analysis, which evaluates post-conditions of actions (what will happen after the action is executed) against the goal.

Data representation

Deliberative architectures are characterised by the usage of symbolic expressions that represent the environment, the robot's actions and the interactions among these two. In order to generate this kind of representations the agent must incorporate the means to transform its sensory information into abstract symbolic expressions. Usually, the symbolic expressions used by the agent are pre-defined by the designer. Defining abstract symbols in this manner incurs in the well known problem of *symbol grounding* [Harnad, 1990], which refers to how the agent can relate the defined abstract symbols to the concrete information it possesses. In other words, given the defined symbols, how can these be instantiated using sensor information?

Action selection mechanism

Deliberative architectures use planning techniques as their action selection mechanism. Planning is used to operate on the symbolic expressions and

elaborate plans, which will dictate the agent's behaviour.

Real-time actuation

Planning is a computationally-intensive, hence a time-consuming task; therefore it proves inappropriate for real-time actuation where actions must be selected rapidly. Moreover, the dynamics and uncertainty of robotic environments pose greater difficulties, as plans need to be constantly updated and re-evaluated to remain sound with the current state of the environment.

Goal-directed behaviour

Deliberative architectures are goal-directed, as the plans are computed for the goal to be achieved. Moreover, these architectures do not need stimuli to produce actions.

Architectural structure

Deliberative architectures are decomposed functionally: this means that their structure follows from the functional decomposition of, first sensing, then planning and finally acting (see Figure 2-1). This functional decomposition does not allow the implementation of different parallel processes to solve the task as the sequential order of sense, plan and act must be retained.

Usage of the agent-environment interaction

Deliberative architectures are provided with *a priori* models of the agent interaction with the environment and use these models to predict the outcome of actions in the future. This is the reason why uncertainty becomes a

problem for the deliberative approach as it can not be modelled *a priori*.

Applicability of learning

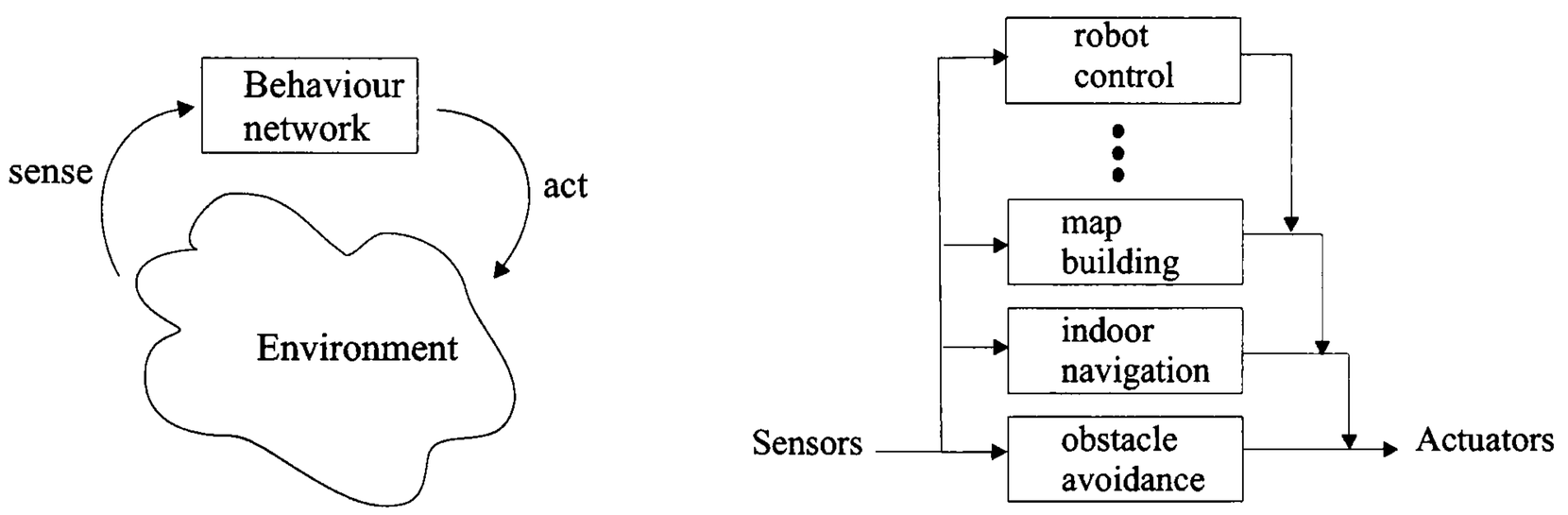
Learning has been exploited in deliberative architectures, for example in learning plans from experience [Carbonell, 1983] or refining planning operators [Carbonell and Gil, 1990].

2.2.2 Reactive or behaviour-based architectures

Reactive architectures, such as: *subsumption* [Brooks, 1985], *situated automata* [Rosenschein and Kaelbling, 1995, Kaelbling and Rosenschein, 1990, Rosenschein and Kaelbling, 1986], *motor schemas* [Arkin, 1989, Arkin, 1987], also known as stimulus-response, embodied or behaviour-based architectures were introduced to overcome the difficulties of applying deliberative architectures in dynamic, complex and uncertain systems, such as robots in their environment.

For deliberative architectures, symbolic expressions are essential for intelligent actuation; in the reactive approach, symbolic representations are eliminated. Reactive architectures are usually defined using a collection of behaviours or *behaviour network* as basic representation elements. Stimuli sensed from the environment are directly introduced into the behaviour network which produce a set of actions that are executed in the environment. Figure 2.2(a) illustrates the general configuration of a reactive architecture. A *behaviour* [Arkin, 1998] is defined as a partial mapping between *stimuli* and *responses*. Stimuli usually relate to the robot's sensory information, while responses relate to the robot's actions. For example, detecting an ob-

stacle is a stimulus, while the corresponding response could be turning to avoid it. Stimulus-response mappings are usually implemented using *if-then* rules. Out of all the collection of behaviours, at any point in time only a sub-set of them are selected for actuation. The mechanism for selecting the appropriate sub-set of behaviours is known as *behaviour arbitration*. Reactive architectures are also known as embodied architectures as the robot's behaviour is the result of the interaction between the robot and the environment [Pfeifer and Scheier, 2001, Steels, 1995c].



(a) Configuration of a reactive architecture (b) Brooks' Subsumption architecture

Figure 2-2: Reactive architecture

One of the first robots based on the reactive architecture was introduced by Grey Walter [Walter, 1953]; the "Machina speculatrix" was a simple autonomous mobile robot, which used a hardware electronic circuit for its control. Light-following, battery-recharging and object-avoidance, were some of its possible behaviours. One of the most interesting characteristics of this architecture was that the robots did not create, store or compute any explicit symbolic representation, and actions were not planned. Instead, the robot's resulting actions were an emergent property of the interaction between the robot, its environment and the stimulus-response behaviours.

Braitenberg's [Braitenberg, 1986] vehicles (autonomous mobile robots) use either direct coupling between sensors and actuators, or simple neural networks, to achieve behaviours such as object-attraction/repulsion, and even capabilities of learning and memorising. Again, these vehicles, did not create, store or compute any symbolic representations.

Brooks [Brooks, 1990, Brooks, 1985] presented the *subsumption architecture*, one of the best-known reactive architectures. The subsumption architecture is a layered architecture, in which each layer corresponds to a stimulus-response behaviour. Simple behaviours are at the lower-end of the architecture, while more complex are at higher-levels (see Figure 2.2(b)). Behaviours do not use any kind of symbolic representation, and do not use planning to decide which actions to select. In this architecture, the robot's global behaviour emerges from interaction between the robot, the network of behaviours and the environment. The subsumption architecture takes its name from the mechanism used for behaviour arbitration. Subsumption allows active complex behaviours to subsume simpler ones. Because of the layered ordering of behaviours, the subsumption architecture allows for incremental development, that is, after building and successfully testing some behaviours, more complex ones can be developed on top of the "working" system. Some authors [Brooks, 1991] have taken the view that this architecture is capable of attaining general intelligence.

Data representation

Reactive architectures do not use symbolic representations; instead they use the input information directly from sensors, alleviating one of the difficulties

of the deliberative approaches, i.e. there is no symbol grounding problem as no symbols are used by this architecture.

Reactive architectures use only implicit representation by allowing behaviours to store and use state representations [Mataric, 2001, Mataric, 1999, Mataric, 1997]. For example, in [Michaud and Mataric, 1999] a tree-like representation is used to store the history of behaviour transitions; this tree is then used to learn and select the most adequate sequence of behaviours. In [Goldberg and Mataric, 1999] *augmented Markov models* are presented and used for modelling the dynamics of the robot-environment interaction. These models are essentially Markov chains with added statistical measures for state transitions. A hierarchical behaviour-based architecture is presented in [Nicolescu and Mataric, 2002]; this architecture introduces abstract behaviours as explicit representations of the behaviours' pre- and post-conditions. Then, networks of abstract behaviours are used to specify plans. In [Mataric, 1992, Steels, 1995c] a subsumption architecture is used for the navigation of a mobile robot, detecting landmarks and build maps which are a representation of the environment.

Action selection mechanism

Action selection is tightly coupled to the sensory information through the stimulus-response behaviour. That is, a sensory stimulus is received and is mapped, using the behaviour network, onto motor outputs.

Real-time actuation

The stimulus-response mechanism produces a fast and computationally inexpensive actuation. This makes reactive approaches suitable for acting in real-time.

Goal-directed behaviour

Most reactive architectures need stimuli to produce motor-outputs, thus the robot's actions are the result of the interaction between the sensory input and the robot's behaviour network. This approach needs sensory stimuli to produce actions, thus it is considered purely reactive. Some alternatives to the necessity of stimuli to produce actions have also been studied. For instance, Maes [Maes, 1989, Maes, 1990] developed a network of behaviour-like elements known as *behaviour networks*, where each behaviour in the network has associated pre- and post- conditions from and to other behaviours. These networks represent the goals for the agent, and by spreading activation through the network, the agent achieves some planning-like capabilities. Using of these types of networks results in reactive architectures being capable of achieving goal-directed behaviour.

Architectural structure

Reactive architectures are behaviourally decomposed (see Figure 2-2) into layeres or modules, each corresponding to an independent and complete behaviour. Here, independent means that the behaviour can work in isolation of other behaviours, and completeness refers to the behaviour taking sensory inputs and producing motor outputs (from input to output). These charac-

teristics of behaviours facilitate their implementation as parallel processes, as each process can be implemented by a behaviour. Moreover, modularity and incremental implementation are also possible using behaviours.

Usage of the agent-environment interaction

The interaction between the agent and the environment is essential in reactive approaches as it drives the agent towards the desired goal. This means that without any environmental stimulus the agent would not be capable of performing any task, as stimuli are necessary for the responses to be activated. Reactive architectures encode the solution to the task within the structure of the architecture, which makes it difficult to achieve flexible goal selection, although some mechanisms can be used to achieve goal-directed behaviour [Maes, 1989, Maes, 1990].

Applicability of learning

Learning has been applied extensively to these architectures. Reactive architectures approach learning by using behaviours as substrate elements for learning [Mataric, 2001]. This means, that the agent must learn which is the appropriate behaviour to execute at any given time. For example, in [Mahadevan and Conell, 1991] a box-pushing task is learned by associating different stimulus-response behaviours to the state observed by the robot. Similarly, in [Maes and Brooks, 1990] a walking task is acquired by learning to coordinate different behaviours.

2.2.3 Hybrid architectures

Hybrid architectures are mostly layered architectures that combine aspects of the, previously seen, reactive and deliberative architectures [Gat, 1998]. Hybrid architectures try to benefit from the fast and computationally cheap response of reactive systems for situations in which there is no time for deliberation, while they can also benefit from deliberation to plan long-term strategies. Figure 2-3 illustrates a generic configuration of a hybrid architecture.

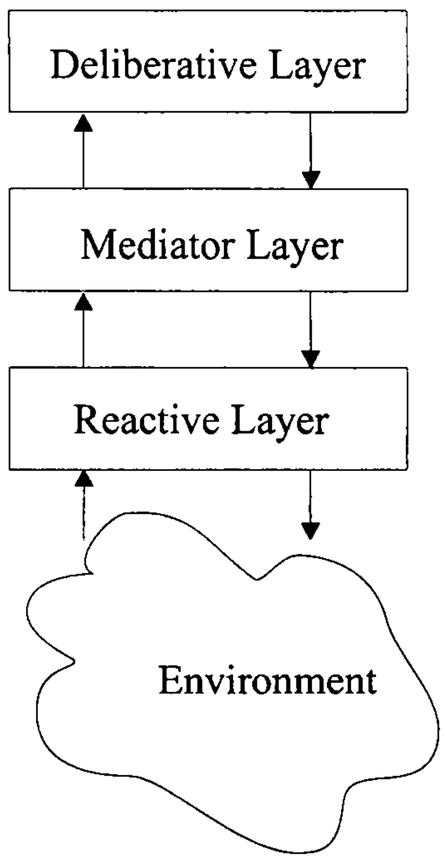


Figure 2-3: Configuration of a generic hybrid architecture

Hybrid architectures have their functionality divided into layers, i.e. a layer for reactive control, a layer for deliberative control and an intermediate layer. Each layer is explained as follows.

The *reactive layer* is in charge of time-critical behaviours. Time-critical behaviours are those that need to be executed in real-time (see Section 2.1.1). For example, in robotics these behaviours could include *object avoidance* or

goal-keeping which must be executed in real-time in order to avoid collisions or letting in a goal. In order to achieve real-time actuation, hybrid architectures use mainly stimulus-response behaviours in their reactive layer. For instance, *SSS* [Connell, 1992] uses PID controllers, which are functions that map sensed input values into an output control value.

The *deliberative layer* deals with the behaviours that need planning, i.e. deliberative behaviours. Deliberative behaviours are used to solve the agent's long-term goals. These could include strategic decision making, such as the strategies used by the robots in a football team to choose the appropriate actions to score goals. The deliberative layers are usually implemented using planning techniques similar to those used for deliberative architectures. For example, *TouringMachines* [Ferguson, 1992] uses a hierarchical planner.

The *mediator layer* has the task of mediating among the reactive and the deliberative layers. Mediating among layers means that, as the agent uses its reactive layer to react to the environment's stimulus, it must also act towards satisfying the plan produced by the deliberative layer. The mediator must be designed taking into account whether plans are first elaborated and then sent to the mediator to be executed such as in *3T* [Bonasso *et al.*, 1997], or whether the mediator requests a plan from the deliberative layer such as in *ATLANTIS* [Gat, 1992].

Other examples of hybrid architectures include: *Procedural Reasoning System* [Georgeff and Lansky, 1987, Ingrand *et al.*, 1992], and Chella's architecture [Chella *et al.*, 1998, Chella *et al.*, 1997b, Chella *et al.*, 1997a] based on the *linguistic, conceptual* and *sub-conceptual* components.

Data representation

Hybrid architectures exploit both symbolic representations, and raw sensor and motor information. This is achieved by dedicated layers which deal with the different types of information. The low levels or reactive layers use raw sensor and motor data just like reactive architectures, while the deliberative layer usually deals with symbolic representations, as in the deliberative approach. In hybrid architectures the symbol grounding problem is addressed by the mediator layer, which must translate the deliberative's layer symbolic information into information understandable by the reactive layer and vice versa.

Action selection mechanism

Action selection in hybrid architectures is more complicated than in reactive architectures, as it needs to take into account the actions selected by the reactive and deliberative layers. The mediator layer is usually in charge of mediating between the two layers, and therefore, coordinating the action selection process.

Real-time actuation

Real-time actuation is achieved in hybrid architectures through the reactive layer, although the actuation of this layer could be slower than in the reactive architecture, given the possible conflicts between reactive and deliberative layers.

Goal-directed behaviour

Hybrid architectures achieve goal-directed behaviour by following the plans generated by the deliberative layer.

Architectural structure

Hybrid architectures are composed of three vertical layers. Each layer of the architecture has different functionality associated with it, namely: react, mediate or deliberate. This layered configuration allows the execution in parallel of the functions of each layer. For example, the reactive layer can be executing low-level navigational behaviours such as *object avoidance* while the deliberative layer could be *path planning*.

Usage of the agent-environment interaction

The interactions with the environment need to be carefully studied, as each layer of the architecture will respond to them in a different manner. The global outcome of the agent decisions will therefore need to be a coordinated response to these interactions.

Applicability of learning

Hybrid architectures have been applied in combination with learning methods. For example, [Benson and Nilsson, 1995] uses a hybrid architecture to learn models (effects) of the actions of an agent. In [Hu and Gu, 2005] a hybrid architecture which uses reinforcement learning to learn fuzzy rules and genetic algorithms to tune its parameters is presented.

2.2.4 Discussion

The previous sections have described the characteristics of the different architectures, these are now summarised in Table 2.1.

Table 2.1: Architecture characteristics

	Deliberative	Reactive	Hybrid
Representation	symbolic	implicit	symbolic
Action selection	planning	stimulus-response	planning + stimulus-response
Real-time action	no	yes	yes
Goal-directed	yes	yes	yes
Structure	functional	behavioural	functional-behavioural
Embodied	no	yes	yes
Learning	yes	yes	yes

In robotics, the characteristic of *real-time* actuation is of central importance. As illustrated in Table 2.1, reactive or behaviour-based and hybrid architectures are capable of real-time actuation, this is because they use reactive action selection mechanisms such as stimulus-response. Contrarily, planning methods are not well suited for dynamic and uncertain environments such as those observed in robotics. The architecture proposed in this thesis uses reactive action selection mechanisms, thus being capable of real-time actuation.

Table 2.1, shows that all of the architectures are *goal-directed*, i.e., all drive the robot towards achieving a goal. Deliberative approaches do so by computing plans that satisfy the goal, whereas reactive approaches 'encode' the goal-directed behaviour in the hierarchy or network of behaviours. That is, the interactions between the robot, the environment and the behaviours, drive the robot towards achieving the goal of the task. This approach re-

sults in a more 'hard-wired' goal-directed behaviour, as changing the goal means having to change the way behaviours are chosen (behaviour arbitration mechanism). Contrarily, planning is more flexible for changing goals, as only the plan needs to be changed. As will be discussed in this thesis, the proposed architecture yields new insights into goal-directed behaviour within the reactive action selection approach.

The structure of any architecture conditions its processing. For example, the functional composition of the deliberative approach conditions it to a sequential processing, i.e. sense, plan and act. Contrarily, reactive and hybrid architectures have a behavioural structure, i.e. hierarchies of networks of interconnected behaviours. This structure allows the important aspect of *parallel* processing. Parallel processing is essential in the robotics domain in order to achieve real-time actuation [Hu and Brady, 1995, Hu and Brady, 1996]. Behaviourally structured approaches allow behaviours to be implemented in concurrent and distributed modules.

As can be seen from Table 2.1, reactive and hybrid architectures are the most suitable architectures to be applied in robotics as they share the characteristics of real-time actuation, goal-directed behaviour, behavioural decomposition, embodiment and applicability of learning. Although these architectures, are suitable for robotic control, they also have some shortcomings. For instance, hybrid approaches need to have high-level and low-level behaviours coordinated by the mediator layer, which often proves difficult to design. An issue with behaviour-based architectures is whether they can extend to higher-level behaviours such as *communication in natural language* or *episodic memory*. It has been argued that, for achieving these behaviours,

explicit representations are necessary [Steels and Baillie, 2003]. Deliberative architectures use explicit representations (symbols), but this approach has failed due to the difficulty of *grounding* the meaning of *a priori* defined symbols with a robot's sensor and motor data. In this context, an architecture which generates representations in a *bottom-up* fashion is desirable, that is, an architecture that uses sensor-motor data and the robot-environment interaction to drive the generation of explicit representations. This thesis develops an architecture for robots based on the bottom-up generation of explicit and grounded representations known as *concepts*.

In conclusion, a robotic architecture must be: (i) reactive, (ii) flexible at defining goal-directed behaviours, (iii) behaviourally structured, and (iv) capable of generating grounded representations. This thesis proposes an architecture based on the reactive and behaviour-based approaches, which generates grounded representations known as concepts, and uses these to learn control behaviours. As will be explained later in the thesis, concepts have a two-fold function:

- Generalisation.
- Intermediate representation.

In this thesis, generalisation is in order to represent a large number of elements using a smaller representation, and creating intermediate representations is in order to bootstrap low-level sensor-motor data into abstract representations with emergent properties.

2.3 Learning and reinforcement learning

As introduced in Section 2.1.1 adaptability is a key factor for achieving autonomy in artificial systems. Moreover, the ability of robots to learn provides them with means for adapting to changing circumstances in their environment [Brooks and Mataric, 1993]. Let us exemplify these characteristics of learning systems with the following example. A robot soccer player has been programmed to successfully (performance measure is high) score-goals (task) using its shooting device. Let us assume that during a football game, this device has been deformed, and therefore its success has been affected (performance measure decreases). It would be useful if this robot had the capability of using the experience with this new shooting device (deformed) to learn a new 'way' of shooting, such that scoring becomes again successful (performance measure increases). In this example both of the previous characteristics of learning systems are represented, namely, the agent autonomously aims at satisfying its goals (i.e. scoring) and the agent adapts to a changing circumstance of the environment (deformation in shooting device).

This section provides a review on the literature related to robotic learning and adaptability. It emphasises especially the reinforcement learning framework, as this sets the context for the learning architecture developed in this thesis.

2.3.1 Introduction

The general definition of an agent that learns from experience is as follows: an agent is said to learn from experience with respect to some tasks or performance measure, if its performance measure for the tasks improves with

experience [Michell, 1997].

In general, any artificial learning system is composed of the following parts.

- **Training experience** this is the data used by the learner to improve its performance at the given task. Depending on the type of data provided, the learner can be classified as a *supervised* or *unsupervised* learner. A supervised learner needs to be presented with the training data and its 'correct' relationship to the task. For example, the training data presented to a neural-network trained using backpropagation is supervised, as the network is presented with the desired output (correct output) for each input [Michell, 1997]. Unsupervised learners do not need to be given the relationship between the data and the task. For example, training a robot to navigate using rewards is an unsupervised learning task, as the rewards can indicate that some actions are bad (e.g. actions that lead to bumping into obstacles) and others are good (e.g. actions that lead to the goal position), but does not indicate which is the appropriate action for the robot to take; it only provides an evaluation measurement (reward). As will be shown later, reinforcement learning is an instance of unsupervised learning.

- **Target function** the *target function* is the function the learner must acquire, i.e. what must be learned. For example, in the previous navigation task, the target function is to navigate to a destination without colliding with obstacles.

- **Representation of the target function** the agent can encode internally the target function using different representations. For example,

a target function could be represented by the weights, thresholds and connections of a neural network, or more simply, in a table representing the output corresponding to each input.

- **Function approximation algorithm** in order to learn the target function, the agent must approximate its representation of the target function towards the desired target function, in other words, to manipulate the representation of the target function so that it approximates to the target function. This process depends on how the target function is represented and the algorithm used to approximate it. For example, if the target function is represented using a neural network, the most likely parameters to be learned are the weights associated with each neuron, and these could be learned using an *approximation algorithm* such as backpropagation.

In robotics it would be desirable to indicate only the goal to be achieved, and for the robot to learn how to achieve it. In some sense, this resembles unsupervised learning, where goals can be defined in terms of rewards, and where the learner can use trial-and-error techniques to gain the highest reward. Moreover, unsupervised learners are more autonomous as they can continually learn by interacting with their environment. Learning in this fashion is the central aim of reinforcement learning, where the learning agent learns through being driven by rewards and trial-and-error. The following section gives a review of the characteristics of reinforcement learning most related to the thesis.

2.3.2 Reinforcement learning

Reinforcement Learning (RL) is the problem faced by an agent that learns, using only the interaction with the environment as training experience, which are the actions to take in order to maximise a performance measure known as reward [Ribeiro, 2002, Sutton and Barto, 1998, Kaelbling *et al.*, 1996].

The framework to solve the RL problem is defined by the following elements [Kaelbling *et al.*, 1996]:

- A discrete set of environmental states, S.
- A discrete set of actions available to the agent, A.
- A set of scalar reinforcement signals (rewards), R.

In robotics, any state, $s \in S$, is defined by the combination of the robot's sensors. For example, if $x_1, x_2, \ldots x_n$ are n sensors, then the state is defined as: $s = \{x_1, x_2, \ldots x_n\}$. Each of the sensors composing the state are known as *state variables*. Each action, $a \in A$, is a possible command that the robot can execute, for example, by setting the speed of its motors. Finally, $r \in R$, is a reward value. Figure 2-4 illustrates the interaction of these elements under the RL framework.

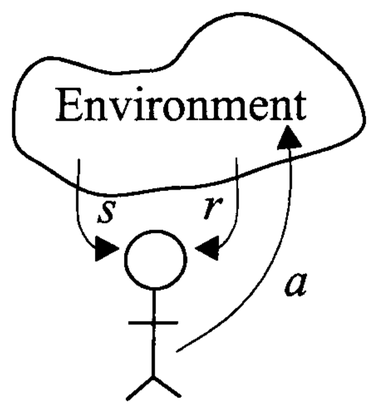


Figure 2-4: Elements of the reinforcement learning framework

Assuming a discrete definition of time, states, actions and rewards can be sequentially ordered to describe the dynamics of any discrete-time system. For example, Figure 2-5 illustrates the dynamics of an agent interacting with its environment. In these dynamics, s_i represents the environmental state observed at time i, a_i is the action executed after observing the state. The effect of the action makes the state change from s_i to s_{i+1}. This state transition is rewarded by r_{i+1}.

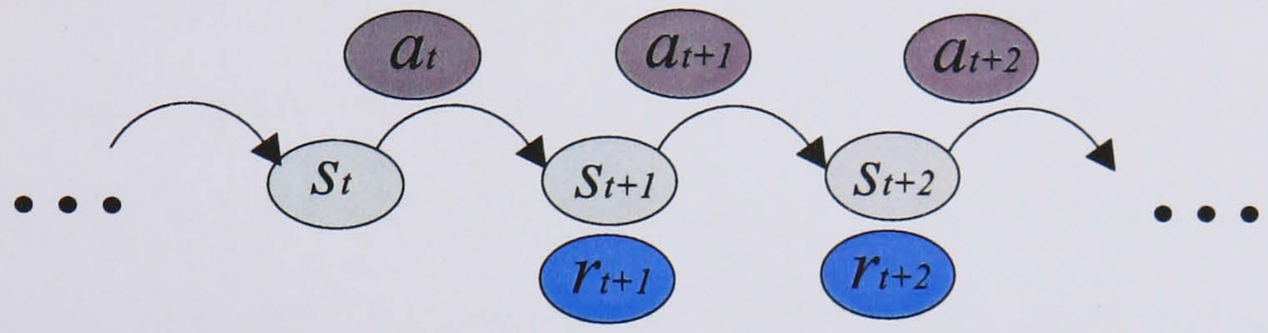


Figure 2-5: Discrete dynamics of state, action and reward

In the RL framework, the agent learns the target function by sequentially and iteratively interacting with the environment as following:

- The agent observes the state of the environment at time t that is, s_t.
- Given s_t, the agent can select and execute an action, a_t.
- After a_t is executed, the agent receives the immediate reward r_{t+1}, for the state transition from s_t to s_{t+1}.
- Finally, the agent uses the training experience represented by the tuple, $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$, to learn the target function.

As can be seen in the above, the RL learner only receives rewards from the environment as feedback. This feedback, in the form of rewards, is known as

evaluative feedback [Sutton and Barto, 1998]. *Evaluative feedback* provides numerical information of how 'good' each state transition was, and thus it does not indicate which was the correct action to execute. Therefore, in order to discover the most appropriate action to execute at any state, the agent will have to iteratively select different actions and learn their outcome through trial and error. From the definitions in Section 2.3.1, it can be said that RL is an unsupervised type of learning, which uses evaluative feedback and interaction with the environment to learn the target function.

As will be seen later, there are various ways to represent the target function. In RL the representation of the target function is commonly known as *policy*. More precisely, a policy is a function that takes the state of the environment, s, as input, and indicates the probability of executing any of the agent's actions a.

RL assumes that the target function is learned when the policy maximises the rewards received over extended periods of time. In other words, an agent is considered to have learned if its policy maximises the reward received over time. Policies that maximise reward over time are known as *optimal policies*.

As seen above, optimal policies are defined using the rewards received over extended periods of time. The reason for using extended periods of time, rather than using the immediate reward is explained in the following example. In a game of chess, taking an opponent's piece could have a high immediate reward, as it has short term advantage to take those pieces. But professional chess-players do not limit themselves to taking the opponent's pieces, as setting the board in an advantageous configuration (long term advantage) can be more important to win the game. A similar reasoning is

carried out when defining optimal policies over extended periods of time, i.e. maximising the immediate reward does not necessarily maximise the reward in the long term. In RL a common formal definition of the reward is the following infinite discounted reward sum:

$$z_t^{\infty} = \sum_{k=0}^{\infty} \gamma^k r_{t+k} = r_t + \gamma^1 r_{t+1} + \gamma^2 r_{t+2} + \ldots \tag{2.1}$$

where, $0 \leq \gamma \leq 1$, is the *discount factor*. If $\gamma < 1$ then the importance of the rewards received in the future are discounted by a factor of γ^k, where k is the step number. Using discounting is equivalent to giving more importance to the rewards the agent can achieve in the near future rather than in the distant future.

The next section reviews some of the algorithms and techniques applied to solve RL problems, i.e. algorithms that maximise the reward represented in Equation 2.1.

2.3.3 Solving reinforcement learning problems

This section reviews some of the popular techniques and algorithms used to solve RL problems. Firstly, *value functions* are introduced as means to evaluate the performance of a policy. Secondly, the relationship between optimal value functions and optimal policies is shown. Finally, some of the methods to find optimal policies using value functions are described.

Value functions

A *value function* is a function that measures how much reward an agent can gain when it acts following a policy, π. In other words, a value function

measures the 'goodness', in terms of reward, of a policy.

The techniques to solve RL problems evaluate policies using mainly two types of value functions, namely *state-value functions* (V^π) and *action-value functions* (Q^π). The following expressions define these value functions:

$$V^\pi(s) = E_\pi\{z_t^\infty \mid s_t = s\} = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid s_t = s\} \tag{2.2}$$

$$Q^\pi(s,a) = E_\pi\{z_t^\infty \mid s_t = s, a_t = a\} = E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \mid s_t = s, a_t = a\} \tag{2.3}$$

The V^π function measures the infinite expected reward (Equation 2.1), $E_\pi\{\}$, of an agent following policy π, and currently being at state, s_t. Similarly, Q^π, measures the infinite expected reward of an agent following policy π, and currently being at state s_t, and having selected action a_t.

Table 2.2: Value function representation

s	$V^\pi(s)$
s_1	$V^\pi(s_1)$
s_2	$V^\pi(s_2)$
s_3	$V^\pi(s_3)$
$\vdots$	$\vdots$

$Q^\pi(s,a)$	a_1	a_2	a_3	$\ldots$
s_1	$Q^\pi(s_1,a_1)$	$Q^\pi(s_1,a_2)$	$Q^\pi(s_1,a_3)$	$\ldots$
s_2	$Q^\pi(s_2,a_1)$	$Q^\pi(s_2,a_2)$	$Q^\pi(s_2,a_3)$	$\ldots$
s_3	$Q^\pi(s_3,a_1)$	$Q^\pi(s_3,a_2)$	$Q^\pi(s_3,a_3)$	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

A simple way to store value functions is by using tables. These tables have one entry for each state s, or state-action pair (s,a), and one output corresponding to the expected reward value of the entry. Table 2.2 illustrates both a state and an action value tables.

Optimal value functions and optimal policies

As stated earlier in Section 2.3.2, RL agents search for an optimal policy, such that it maximises the reward in Equation 2.1. Let us denote an optimal policy by π^*.

An *optimal policy* can be defined as a policy where its value function is maximum for all the possible states and actions when compared with any other policy π; in other words: $V^{\pi^*}(s) \geq V^{\pi}(s)$ for all $s \in S$ and $Q^{\pi^*}(s,a) \geq Q^{\pi}(s,a)$ for all $s \in S$ and $a \in A$. The value function of an optimal policy is known as *optimal value function.*

By definition, finding optimal value functions implies finding optimal policies, thus an agent can learn optimal policies by searching for the optimal value functions, i.e. functions that maximise long term reward. The following sections review how to learn optimal policies by computing optimal value functions.

Finding optimal policies using dynamic programming

Dynamic programming techniques, such as *policy iteration* and *value iteration* [Bellman, 1957] are methods that exploit *Markov Decision Processes* (MDP) and the *recursive Bellman optimality equations* to calculate optimal policies.

Bellman's recursive optimality equations are defined using what is known as the *agent-environment models.* These models are: the *one step transition probability* $P^a_{ss'}$, and the *expected immediate reward*, $R^a_{ss'}$. The one step transition probability indicates the probability of observing state s', given that the current state is s and the agent performs action a. In other words,

the model indicates, in a probabilistic manner, the future state when an agent chooses an action, in a given state. Having this model is equivalent of knowing *a priori* the dynamics of the agent-environment interaction. The expected immediate reward model indicates the reward expected for selecting action a, at state s, given that the future state will be s'. Having this model is equivalent to knowing the rewards the agent will receive in the interaction with the environment.

The requirement of these models to compute optimal policies restricts the usage of dynamic programming techniques in robotics, as most of the times, models are not available *a priori*. Moreover, these techniques are computationally expensive, which makes them inappropriate for robotics, where limited computational power is a constraint. For these reasons, dynamic programming techniques are not discussed in this thesis any further.

Finding optimal policies using temporal difference learning

Given the shortcomings of dynamic programming techniques, temporal difference methods are introduced. Temporal difference methods [Sutton, 1988] are capable of solving the RL problem without relying on world models ($P^a_{ss'}$ and $R^a_{ss'}$); instead these techniques use the agent-environment interaction to acquire all the necessary information. Temporal difference methods rely on what is known as *bootstrapping* to update the value functions [Sutton and Barto, 1998].

One of the best known temporal difference method is Watkins' *Q-learning* algorithm [Wactkins and Dayan, 1992, Wactkins, 1989]. The reason for its popularity relies in its conceptual and implementation simplicity.

Q-learning learns optimal policies by finding optimal q-value functions. The optimal q-value function is found by iteratively applying the following update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \, max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

where s_t is the active state, a_t is the selected action, α and γ are learning parameters, r_{t+1} is the reward received for the transition between s_t and new state s_{t+1}, and $Q(s_t, a_t)$ is the q-value function.

Finally, after learning the q-value function, finding the optimal policy π^* is achieved by finding the action that maximises the q-value function at the given state, as follows:

$$\pi^*(s) = arg \; max_a Q(s, a)$$

where $arg \; max_a$ denotes the value of a at which the expression that follows is maximised.

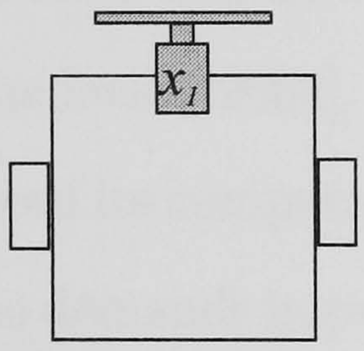
Other temporal difference methods to solve the RL problem include *Actor-critic methods* [Barto *et al.*, 1983], *TD(λ)* [Sutton, 1988], *Dyna* [Sutton, 1991, Sutton, 1990], *Prioritized Sweeping* [Moore and Atkeson, 1993] and *Queue-Dyna* [Peng and Williams, 1993].

2.4 Practical reinforcement learning

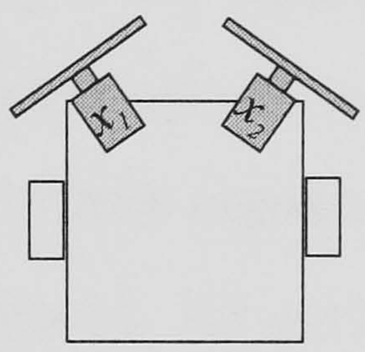
The previous section has reviewed reinforcement learning from a theoretical viewpoint. This section reviews the issues that emerge when applying reinforcement learning in robotics.

2.4.1 Dimensionality problems in learning

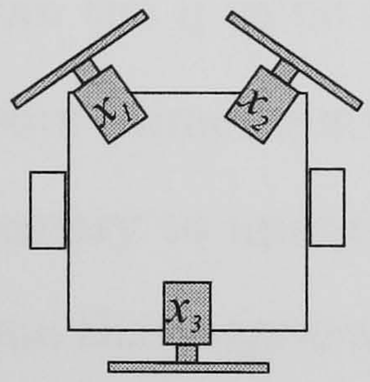
The *dimensionality* problem relates to the size of the robot's state and action spaces, represented as $|S|$ and $|A|$. Let us describe the dimensionality problem with the following example.



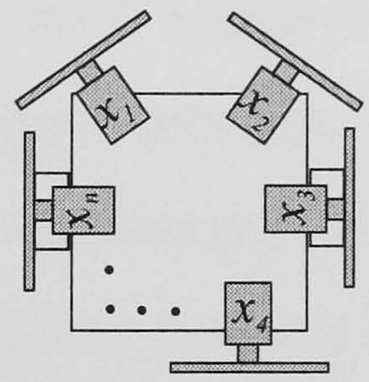
(a) One sensor $|S| = 2$



(b) Two sensors $|S| = 4$



(c) Three sensors $|S| = 8$



(d) n sensors $|S| = 2^n$

Figure 2-6: Exponential growth of the state space size

Figure 2.6(a) illustrates a mobile robot with a single bumper sensor, x_1. The robot's state space, S, is determined by the values of sensor x_1, that is: $S = \{x_1\}$. The size of the state space is $|S| = 2$, as x_1 only has two states, i.e. $s(x_1) = 1$ (pressed) or $s(x_1) = 0$ (released). Figure 2.6(b) illustrates the same robot with an added sensor s_2. In this case, the state space is formed by the combination of the two sensors $S = \{x_1, x_2\}$. The size of the state space composed by two sensors is $|S| = 4$, i.e. the following states: $s = (0, 0)$, $s = (0, 1)$, $s = (1, 0)$ and $s = (1, 1)$. The size of S for the robot in Figure 2.6(c) is $|S| = 8$. As can be seen, the size of a state space grows exponentially

with the number of sensors, in this particular example, of the order of 2^n for n binary state sensors as illustrated in Figure 2.6(d).

The exponential growth of a hyperspace (e.g. the state space above) as a function of its dimensions (e.g. robot sensors) is known as the *curse of dimensionality* [Sutton and Barto, 1998, Kaelbling *et al.*, 1996, Bellman, 1957, Barto and Mahadevan, 2003]. That is, any system operating on such a hyperspace will need its computational resources to grow exponentially in order to cope with the demands imposed by the hyperspace. For example, if a robot is using Q-learning to solve an RL problem, it will need a memory of size $|S| \times |A|$ to store the q-value function. As $|S|$ grows exponentially with the number of sensors (dimensions), so do the memory demands. Moreover, the experience necessary to update a q-value function also grows exponentially.

Another issue that aggravates the dimensionality problem in robotic systems is related to the nature of sensors and actuators, i.e. because these provide continuous responses [Sutton, 1996, Smart and Kaelbling, 2002]. For example, if the bumpers in Figure 2-6 are replaced by luminosity sensors, which are assumed to return a value in the range of 0 to 255 proportional to the environmental light conditions, then the size of state space becomes $|S| = 255^n$, for n luminosity sensors. In general, the size of a state space of a robot can be determined by, $|S| = \prod_{i=1}^{i=n} p_i$, where n is the number of sensors and p_i is the range of values of the i-th sensor.

In order to deal with the problems that arise from the dimensionality of large state and action spaces, algorithms to solve RL problems incorporate what is known as *generalisation* methods. The following section reviews some of these methods.

2.4.2 Generalisation in reinforcement learning

One of the elements of any learner system is the target function representation (see Section 2.3.1), which in RL is known as a policy. In Section 2.3.3) we showed how policies can be defined using value functions ($V^{\pi}(s)$ and $Q^{\pi}(s,a)$) which are computed and stored in table-like structures (see Table 2.2).

Tabular representations of value functions and thus definition of policies are not practical when state and action spaces are large, e.g. containing thousands or maybe millions of state-action pairs. The impracticality lies in the memory requirements and the inefficient usage of training experience to update these functions. To overcome these impracticalities, generalisation methods are applied to solve RL problems.

Generalisation in RL aims at: (i) representing the value functions as compactly as possible and (ii) using past experience to infer information about unseen situations. In some sense, this is equivalent to 'making the most' of the information gathered previously. This section reviews two different methods used to generalise value functions, namely, *function approximation* and *variable resolution*. Following is a review of these two methods.

Generalisation using function approximation

Function approximation techniques are an instance of inductive learning, which hypothesises that any function found to approximate a target function over a large set of training points will also approximate unobserved instances of the target function [Michell, 1997]. In other words, gathering partial information of a function's values can be used to guess the remaining

values of the same function. Function approximation techniques can be used to generalise the value functions used in RL.

In general, function approximators operate as illustrated in the following example. Let $V(s)$ be a value function to approximate depending on the state s. Let $g(p, s)$ be the approximator, where p is a set of adjustable parameters; g is also dependent on s. Then, the approximator operates according to the following steps:

1. Observation of an instance of the state, e.g. $s = x$.

2. Prediction of the value of $V(x)$ using $g(p, x)$; let the value of the prediction be $\hat{v}$.

3. Observation of the real value of $V(x)$; let this value be v.

4. Parameter update towards minimising the error between v and $\hat{v}$.

After the approximator's parameters have been updated for a sufficient number of instances, the approximator can be used to predict the value of any instance of state, seen or unseen, known as a *query point*. There are various methods for function approximation, using different types of parameters and techniques to adjust parameters. In general, these can be classified into two groups, namely, parametric and non-parametric approximators.

Given a set of n parameters, $P = \{p_1, p_2, ..., p_n\}$, a *non-parametric function approximator* uses a subset $p \in P$, to calculate the value of the approximation. The subset of parameters is selected according to the Euclidean distance between the query point and the function's parameters. Similarly, only the subset of near parameters is updated. The update is achieved by changing the values of the subset of parameters towards the observed value.

If the subset of parameters is not sufficiently near the query point, then new parameters can be added. Non-parametric methods include: *nearest neighbour* which chooses only the closest parameter to the query point for the prediction, *distance weighted averaging* and *locally weighted regression* which assign higher relevance to the parameters closer to the query point, according to their distance [Schaal and Atkeson, 1994, Atkeson *et al.*, 1997b, Atkeson *et al.*, 1997a, Stefan *et al.*, 2000, Smart and Kaelbling, 2000].

Parametric function approximators use the complete set of available parameters to approximate the target functions; these being represented as a vector, $\vec{P} = \langle p_1, p_2, ..., p_n \rangle$. Parametric approximators include: linear approximators, and non-linear approximators, such as *multilayer neural networks*. Linear approximators are probably the generalisation method most commonly used in combination with RL as some theoretical work exists that predicts the effects of incorporating function approximation and RL algorithms [Tsitsiklis and Van Roy, 1996]. A linear approximator tries to predict the outcome of the function to approximate, as a linear combination of some weighted features as shown in the following expression:

$$v = g(\vec{P}, \vec{\Phi}) = p_1\phi_1 + p_2\phi_2 + ... + p_n\phi_n = \sum_{i=1}^{n} p_i\phi_i \tag{2.4}$$

where v is the predicted value, $g()$ is the function approximator, $\vec{P}$ are the parameters of the approximator and $\vec{\Phi}$ are some features extracted from the input. Feature are selected by functions known as *feature selectors*; some of these include: *CMAC* [Albus, 1975], *tile coding* [Sutton and Barto, 1998, Sutton, 1996, Stone and Sutton, 2001, Kuvayev and Sutton, 1997], *radial basis functions* and *kanerva coding* [Kostiadis and Hu, 2001]. CMAC, tile cod-

ing and radial basis functions, use similar methods to select features. In general, and as illustrated in Figure 2-7, these methods divide the environmental state into regularly distributed overlapping areas or features, which become active (i.e. selected for the approximation) if the observed state falls inside their area. Figure 2-7 illustrates a two-dimensional state space (represented in thick black) and two sets of overlapping features (represented in grey scales). The total number of features and parameters in linear approximators is always smaller than the size of the state space, but still proportional to it.

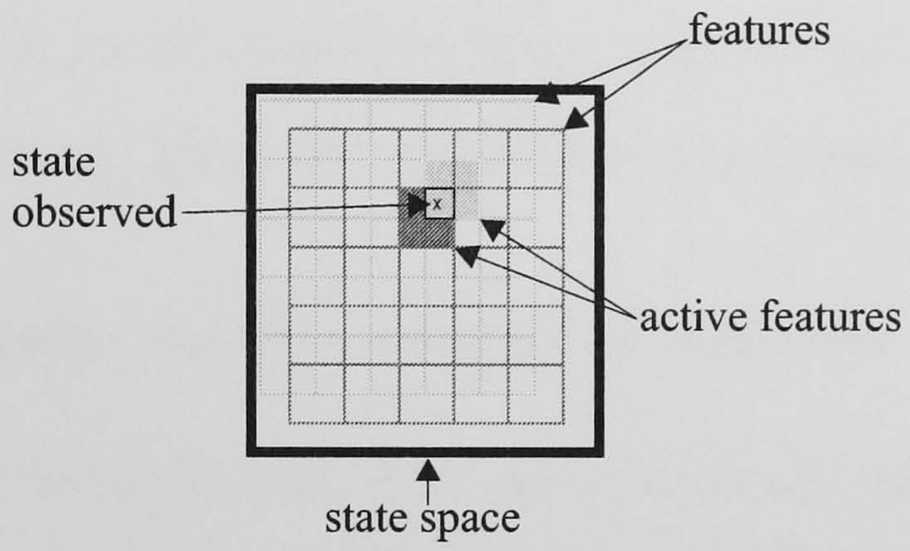


Figure 2-7: CMAC feature selector

Although these function approximators have been successful in reducing the dimensionality of some RL problems, they still have serious limitations. For instance, non-parametric function approximators need a number of parameters proportional to the input's size, thus the curse of dimensionality remains a problem as large state spaces will result in a large number of parameters. Similarly, as parametric approximators use sets of features that are proportional to the dimension of the state space, these will also grow with the size of the state space. Some methods exist to reduce the number of necessary features, for example [Santamaría *et al.*, 1998] uses CMAC with a variable

resolution which assigns higher resolution in important regions of the state space. Another possibility is to use kanerva coding [Kostiadis and Hu, 2001] which provides features based on the function to approximate rather than based on the input state.

In summary, it can be said that parametric and non-parametric approximation methods produce a generalisation of the value function based on fitting parameters, the combination of which approximates to the value function. We refer to this type of generalisation as *implicit generalisation* as the approximation is implicit in the parameters of the approximator. There exist other methods which are not based on fitting parameters, and which produce *explicit generalisations*. These are reviewed in the following section.

Generalisation using variable resolution methods

Variable resolution methods represent value functions similarly to *multi-grid* methods [Chow and Tsitsiklis, 1991, Vollbrecht, 1999]. Multi-grid representations partition the state space into layered grids of uniform resolution. High-level layers are of coarser resolution than lower layers. This layered representation allows one to re-use the value functions learned at high-level layers (coarse resolution) in the lower-level layers (fine resolution).

Figure 2-8 illustrates a generic multi-grid state space representation, where state space is a two-dimensional plane and the states are the grids on the plane; the arrows illustrate that the information learned at high layers is transmitted to lower ones. Although multi-grid representations allow faster learning, as they bootstrap the value function information from high layers to lower layers, they still suffer from the dimensionality problem, as grids

are regularly distributed, thus proportional to the size of the state space. Moreover, having to define the number of layers *a priori*, implies having to find experimentally how many layers (equivalent to the resolution of the representation) are necessary to achieve a desired performance.

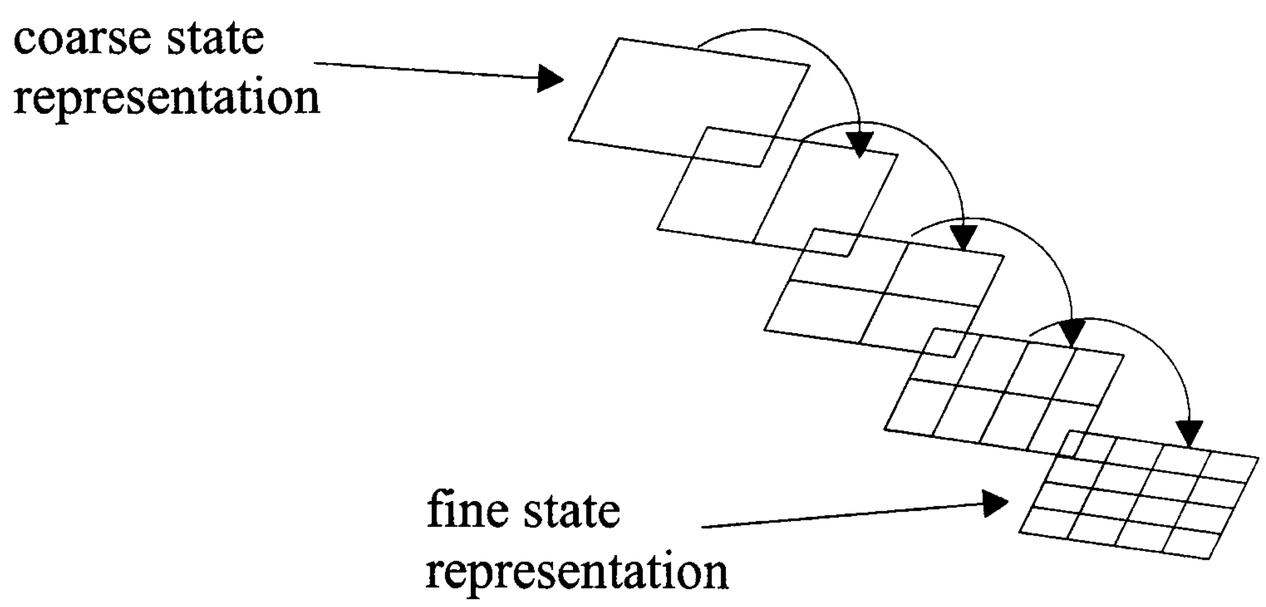


Figure 2-8: Generic multi-grid representation

In order to alleviate these problems, variable resolution methods only represent the areas of the state space that are considered interesting. This results in a method that can deal with dimensionality, as only a portion of the total states are represented.

Variable Resolution Methods [Reynolds, 2000, Munos and Moore, 1999, Moore and Atkeson, 1995, Moore, 1991, Chapman and Kaelbling, 1991] and [Simons *et al.*, 1982] start by representing the state space using a coarse representation, which is partitioned into finer-grained representation during learning. As the representation is acquired on the fly, unlike multi-grid methods, no *a priori* assessment of the necessary resolution must be made. In order to partition only the interesting areas of the coarse state representation, the learner must have the metrics for assessing the interest of the different states, known as *partition metrics*. Different partitioning metrics can be used,

for example, [Reynolds, 2000] uses a partition metric based on the difference between policies, i.e. if the policy changes within the same area of the state space, this area is partitioned. [Simons *et al.*, 1982] uses a measure of the local reward received to partition areas of the state space which receive low reward values. The *parti-game* algorithm [Moore and Atkeson, 1995] assigns a different label to the states which can and can not reach the goal state, then neighbouring states of different class are separated. Others use statistical measures. For instance the *G-tree* method [Chapman and Kaelbling, 1991] uses a T-test to measure the confidence that two areas of the state representation have different reward distributions.

Figure 2-9 illustrates a generic state representation obtained by a variable resolution method. As can be observed, the distribution of states is not uniform through the state space; the arrows indicate that the information of the coarser representations can be bootstrapped into the more refined representations.

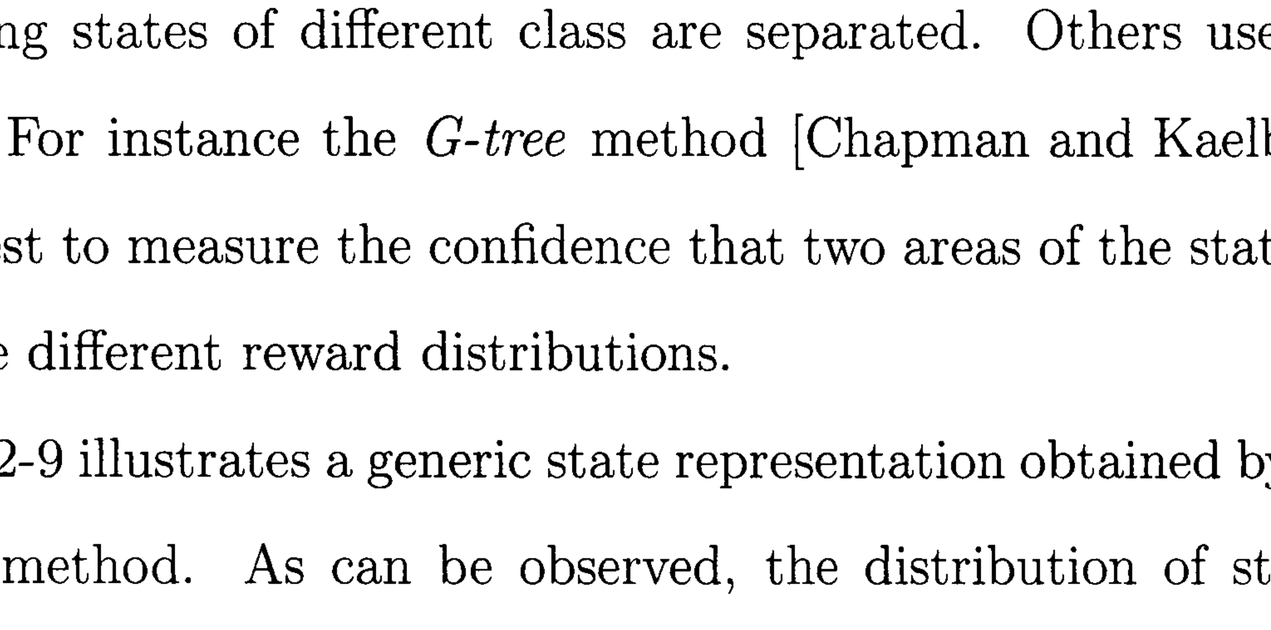


Figure 2-9: Generic variable resolution representation

2.5 Summary

As stated in Chapter 1, this thesis is concerned with designing autonomous robotic agents.

The chapter starts by defining the main characteristics of such agents. Section 2.1.1 shows that adaptability is a main requirement for autonomy. Adaptability can be achieved by adding learning techniques to autonomous agents (see Section 2.3). Thus, designing agents which are capable of learning became the main motivation of this research.

In order to design any system, including autonomous learning agents, there is a need to select an architecture which meets its requirements. To that end, Section 2.2 reviews the main architectures used for the design of autonomous robotic systems. As shown in Section 2.2.4, all of the architectures reviewed can be used in a combination of learning techniques, and the main differences between architectures lies in how suitable they are for operating in robotic environments. The review of robotic architectures shows that *behaviour-based* and *hybrid* architectures are the architectures that have achieved most success in robotic environments. This success is related to the fact that these architectures are capable of achieving *real-time actuation* and flexible *goal-directed behaviour*.

Section 2.3 reviews the literature related to the problem of autonomous agents learning from rewards in their environments, namely the reinforcement learning problem. The conclusion from that section is that there are theoretical proofs that indicate that the reinforcement learning problem can be solved using a range of techniques.

Section 2.4 reviews the practical issues that emerge when applying tech-

niques to solve reinforcement learning problems. Of great importance are issues related to dimensionality (see Section 2.4.1). In order to deal with these issues, Section 2.4.2 introduces generalisation methods. From there it is possible to conclude that function approximators and variable resolution methods do not address the dimensionality problem directly. The main reason is that such methods do not aim to eliminate the irrelevant state variables, but only to construct a compact representation of, possibly irrelevant, state variables.

Hierarchical approaches have been introduced to deal with the irrelevant state variables [Andre and Russell, 2002, Barto and Mahadevan, 2003, Dietterich, 1998]. These methods rely on the designer to determine which are the relevant state variables for each task or sub-task and ignore the non-relevant variables.

This thesis explores a novel method to address the dimensionality problems that arise in RL. Firstly, an architecture for learning and control is proposed that exploits *explicit generalisation*. Secondly, it is shown how the methodology of *Q-analysis* could be used to detect and remove irrelevant state variables.

Chapter 3

Concepts and Concept Generation

The previous chapter introduced robotic architectures and reinforcement learning. An important conclusion from that chapter was that a learner with large state and action spaces requires generalisation methods for:

- Representing the target function in a practical manner.
- Using training experience in an efficient manner, i.e. reducing learning time and generalising to 'unseen' situations.
- Alleviating problems related to the curse of dimensionality.

As seen earlier (Section 2.4.2), generalisation techniques define compact representations of the target function (value function in RL) and use training experience to update large portions of the target function.

This chapter defines concepts as general classes of primitives. Thus, by definition, concepts are a generalisation of the primitives that compose them.

From this definition, it is conceivable to see concepts as generalisers for machine learning problems. This idea of generalisation using concepts is central to this thesis and in the next chapters it will be shown how to define and use concepts to this end.

3.1 Fundamental aspects

3.1.1 What are concepts?

The idea of concept is central to this thesis, since the learning architecture developed in the next chapter uses concepts to learn and to represent the target function. Moreover, concepts will also be used for robot control.

A general definition of a *concept* by the Merriam-Webster Online Dictionary is as follows: (1) Something conceived in the mind. (2) An abstract or generic idea generalised from particular instances [Merriam-Webster, 2004]. The first defines concepts as artifacts built and used in human minds, including those used in natural language communication as words. This thesis does not discuss the concepts used by humans; it only focuses on concepts from an AI perspective, as they apply in robotics. In the second definition, concepts are a generalisation of particular entities or primitives, for example, the concept *transport_vehicle* could be a generalisation of primitives such as, *bicycle, car, bus, train, plane, etc.*

Machine learning approaches include *concept learning* as a particular instance of *inductive learning* [Rendell, 1986, Michell, 1997]. In brief, concept learning is the task of finding general descriptors, known as *hypotheses*, which associate primitives with general concepts. This task can also be seen as one

of *classification* [Chandrasekaran and Goel, 1988], where primitives need to be assigned to general classes. The task of *pattern recognition* can also be seen as a task of finding concepts, as primitives sharing the same pattern are considered as belonging to the same class [Jain *et al.*, 2000].

In this thesis, *concepts* are the result of the classification of primitives into general classes. As concepts are formed by a set of primitives, any concept needs to be associated with a *representative*. For example, in robotics a concept such as 'moving forward' could be composed of all the primitive actions that drive the robot forward. To execute the 'moving forward' concept, one of the many possible primitives needs to be selected and sent to the motors; we call this primitive the representative. Concept representatives will be defined differently depending on the method used to classify primitives. For example, if one is using a clustering technique, then the centres of the clusters can be the representatives of the cluster or concept.

	Shape	**Colour**	**Material**
$ball_1$	round	red	plastic
$chair_1$	non-uniform	black	plastic
$ball_2$	round	blue	leather
$chair_2$	non-uniform	gray	metal
$ball_3$	round	back&white	plastic

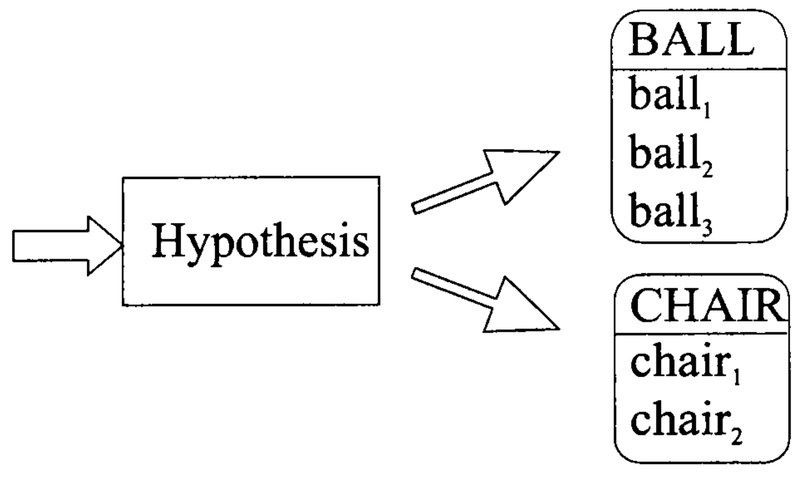


Figure 3-1: An example of a hypothesis in concept learning

The primitives to classify are usually described by means of *properties* here known as *variables*. Then, hypotheses try to classify these primitives, according to their variables, as a corresponding concept. For example, Figure 3-1 illustrates the desired behaviour of a hypothesis, which classifies each

of the primitives, chair_1, chair_2, ball_1, ball_2, ball_3, into the corresponding concepts CHAIR and BALL. In this example, the variables used to describe the primitives are shape, colour and material. Many other different variables could be selected to describe primitives, for example the size and weight could have also been added to the description. In general it is desirable to select the variables that provide the most discrimination between primitives that belong to different concepts, and the less discrimination between primitives belonging to the same concept.

Hypotheses can be defined and represented in various ways. For example, Figure 3-2 illustrates a two-dimensional variable space representing some primitives y. A hypothesis could be used to define regions in the variable space. That is, the hypothesis would indicate that a primitive y_n is classified as belonging to a concept c_m, if the primitive's variables are within the region that belongs to c_m.

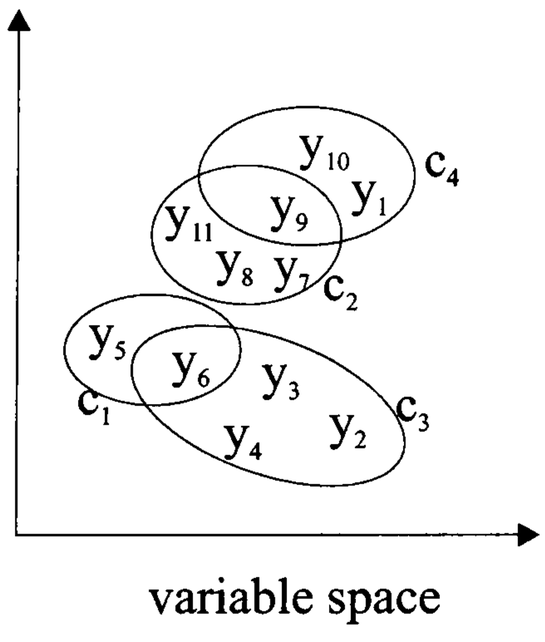


Figure 3-2: A two-dimensional space representing primitives and some concepts

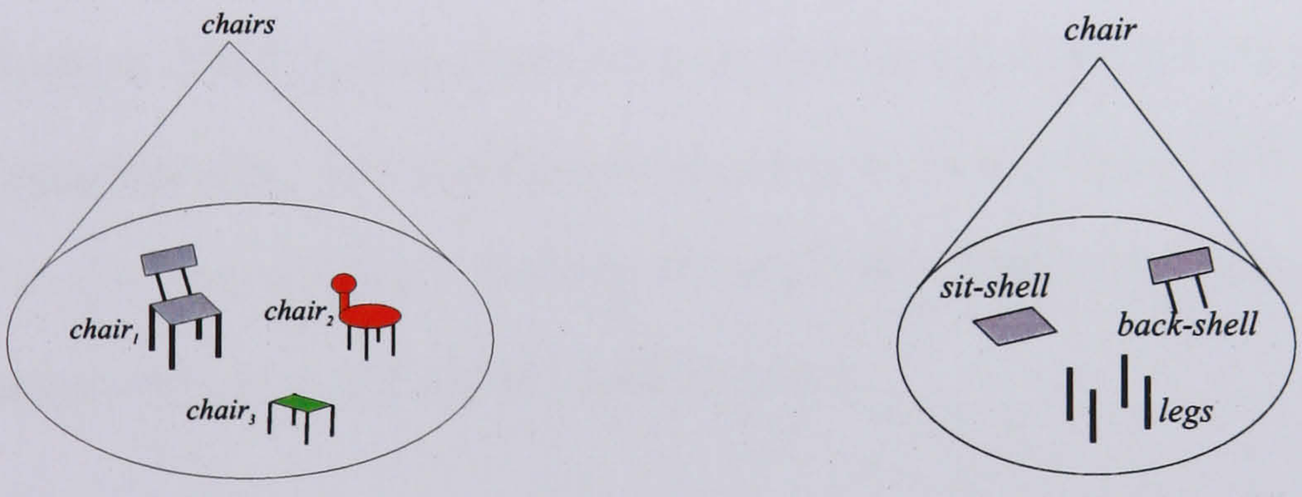
This thesis uses two different methods to represent primitives and hypothesis, one based on distance-based clustering and the other based on

classification using Q-analysis. This chapter describes these different methods. Chapters 5 and 6 present the experimental results of the classification using the two different methods.

The method based on Q-analysis, which is a novel contribution of this thesis, is based on representing primitives as *simplices* and uses their structural properties to define similarity and thus the hypothesis. The following section describes the fundamental differences that emerge in concepts when these are generated using the two different methods.

3.1.2 Sets versus relational structures in classification

During the research reported in this thesis, a major distinction emerged between concepts that represent a class of primitives and concepts that combine primitives.



(a) Set classification.

(b) Relational classification.

Figure 3-3: Set versus relational classification

For example, suppose that one observed three different chairs $chair_1$, $chair_2$ and $chair_3$, then the concept that describes these could be *chairs*. This can be illustrated as in Figure 3.3(a). In this illustration, primitives

constitute the base of a cone, and the concept is the vertex of the cone. A characteristic of this type of concept is that an individual primitive is sufficient to be classified as part of the concept.

On the other hand, the concept *chair*, illustrated in Figure 3.3(b), relates to the set of primitives: sit-shell, back-shell and legs. In this case, all the primitives are necessary for them to be part of the concept *chair*, that is, to have a concept *chair* you need to have a sit-shell, a back-shell and legs, i.e. the relationship between the three primitives is required.

Clearly, these concepts are different. In the first, any primitive is sufficient, in the second all the primitives are necessary to be classified as the concept. In this thesis the first type of concepts are known as *generalisation concepts*, as a set of primitives are generalised into a more compact concept. The second type of concepts are known as *relational concepts*, as they require a relational structure between their primitives, i.e. having the legs under the sit-shell and the back-shell perpendicular to the sit-shell. Elsewhere [Johnson, 1983], generalisation concepts are defined as being the result of an OR-aggregation, and relational concepts as being the result of an AND-aggregation. In Section 4.3 it will be shown how these two types of concepts can be integrated on a multilevel architecture.

3.1.3 Multidimensional data

Multidimensional or multivariable data represent the fact that things are described, related and reasoned about with various dimensions or variables. For example, in order to describe a physical object, the following variables could be used: *size, colour, shape, texture, material composition, odour, etc.*

In diagnosing a patient's disease, one could use the variables *age, sex, temperature, blood pressure, bacterial presence, etc.* to relate the patient to any particular disease. To reason whether to take an umbrella, one could be looking at the variables *cloudy sky, weather forecast, season, etc* to predict if it will rain or not.

Robots also use multidimensional data to describe and reason about their environment. For example, a mobile robot could use the information provided by sonars, thermometers, cameras, bumpers, encoders, compasses, etc. to describe their surroundings and decide where to navigate to. Different sensors provide different kinds of information depending on their physical characteristics. For example, sonar sensors can be used to measure the distance to walls and objects, thus they provide a continuous range of measurement. Bumper sensors provide binary information, either bumped or not, when hitting an obstacle. Compass sensors could provide the direction of the robot based on the polar coordinates, i.e degrees with respect to the north, south, etc. As different types of sensors are added to a robot, it becomes more difficult to interpret the meaning of what they describe. For example, how can the observation of two different sensors be compared? Is a change of one meter in distance the same as one degree centigrade change in temperature? These questions don't have a simple answer. In order to clarify some of the implications of using combinations of different variables or sensors, the next section presents some of the main characteristics of variables and discusses some of the wrong assumptions.

Types of variables

As stated in the previous section, various types of variables exist depending on what and how they represent information. In general, variables can be of the following standard types:

- Nominal: this type represents values using discrete states or labels which have no clear ordering. For example the variable colour could be either *blue, red, yellow, etc.* which have no clear order with respect to each other.

- Ordinal: this type represents values states or labels that follow some order. For example a discrete ordinal variable such as blood pressure could be either *low, medium, high* where the states can be ordered from small to big (low to high). A continuous ordinal variable has ordered continuous measurements, but the scale is not known, it could be exponential, logarithmic, etc.

- Interval: this type represents values on a linear scale which has positive and negative values. This type of variable allows ranking order between measures and also comparison of magnitudes. For example using the variable temperature, it can be said that 30° is less than 40° and that 10° is their difference.

- Ratio: this type represents values in a positive, continuous and non-linear scale. Ratio variables have defined an absolute zero. An example of a ratio variable is the speed of a mobile robot, in which case, one could say that the speed of 1 m/s is as twice as fast as 0.5 m/s.

As mentioned previously, hypotheses assess whether a primitive belongs to a concept based on its descriptive variables. Given that variables can be of the above types, an important issue is to know how variables of different types affect the behaviour of hypotheses. In other words, what are the implications in classification when using *nominal, ordinal, interval* or *ratio* variables in the description of primitives?

If the primitive to be classified is described by nominal variables, one can only decide whether these variables are related or not to some target value. That is, if the variable describing the colour of an object is *green* and is compared with a target colour *red*, then it can be concluded that these two are not related; nothing else can be decided. Similarly, if ordinal variables are used to describe primitives, then one can establish the previous relationship, and it can also say something about ordering. That is, if the variable describing blood pressure of a patient is *low*, then it can be decided that the variable is not related to the target value *medium*, and it can also be decided the following relation $low < medium$, i.e. low is below than medium; but not how much below it is, as their scales are not known. Interval variables take the ordering one step forward; in this case, a primitive described using interval variables can be related to a target value and it is possible to assess its order and measure the difference from the target. That is, a temperature variable being 20 °C can be compared with a target value of 40 °C. We can decide that variable and the target are not related; we can also establish the ordering relation 20 °C < 40 °C, and finally we could say that target value and the variable value are separated by 20 °C. This last information can be extracted as interval variables are defined over a known linear scale.

Finally, if the primitive is described using ratio variables, one could decide their relation to a target, their order, but not their degree of similarity-dissimilarity as their scales are, in principle, non-linear. If the scale of these variables is known, then different transformations of scale could be carried out in order to have a linear representation. In this case, as ratio variables have an absolute zero, it is possible to assess the proportion between the variables.

The implications of describing primitives with different types of variables are significant for the classification task; it is therefore imperative to have a clear description of what each variable represents and what information can be meaningfully extracted from the representation. The view of mapping primitives into multidimensional coordinate spaces and assuming their distance to be a measure of their similarity can only be done if the properties of the coordinate space and its dimensions are known. For example, the distance-temperature coordinate space can define a meaningful similarity between primitives, if and only if the equivalence between a unit of temperature and a unit in distance in relation to the classification are known; and the scales of their dimensions are also known.

3.2 Multidimensional data classification

As explained in the previous subsection, a robot's sensors and actions span multidimensional data spaces. Generally, these spaces are very large and exactly the same combination of measurements is hardly ever observed.

For example a simple robot with a distance, a temperature and two light sensors, each having a 256 values, result in a sensor space of 2^{32} possible state

combinations. The combination (130, 197, 4, 83) could occur on one occasion, and a different combination, (133, 195, 5, 90) might occur on another. Although these are different, for many purposes, they could be considered equivalent. This section introduces some of the existing techniques that aim at finding classes of primitives which can be considered as equivalent. The section concludes by stating the limitations of the reviewed methods, and sets the context for the introduction of a novel approach based on Q-analysis.

3.2.1 Introduction

The body of work in analysis of systems with multiple dimensions or variables is extremely large, including aspects such as: *structural simplification, classification, grouping variables, analysis of dependence and interdependence, hypothesis construction and testing* [Kendall, 1980, Feinstein, 1996]. This section focuses mainly on the methods used for the classification of multidimensional data. In general terms, classification is the task of grouping elements of usually large sets, known as *exemplars* or *instances*, into a smaller set known as *classes* [Chandrasekaran and Goel, 1988, Rendell, 1986], in our terms, grouping primitives into concepts. When classifying multidimensional spaces, a guiding principle is that points that are close are more similar than points that are further apart; thus classes of similar or near points can be formed. In the example above, the two combinations are closer to each other when compared with e.g. the combination (12, 43, 221, 191).

However this idea of similarity depends on the mathematical properties of the multidimensional space that defines each combination. In the first instance, it is conceivable to see the space as an n-dimensional Euclidean

space $\mathbb{R}^n$, with the Pythagorean metric that measures the distances between points $(x_1, x_2, ..., x_n)$ and $(x'_1, x'_2, ..., x'_n)$ as $\sqrt{\sum_{i=1}^{n}(x_i - x'_i)^2}$. However in non-homogeneous multidimensional spaces, how can one be sure that, for instance, a unit of temperature means the same thing as a unit of a light sensor? In other words the scales used and their normalisation have a significant effect on any computation of distances, as is further explained and exemplified in Section 3.3.1.

In a multidimensional context, each primitive, $y \in Y$, to be classified can be described by a set of n variables (see Section 3.1.3). Let us refer to these as *descriptive variables* and denote them by $x = \{x_1, x_2, ..., x_n\}$. A subset of only the relevant variables is selected in order to inform the classification (see Feature Selection Problem in Section 3.3.2). Let us refer to the subset of relevant descriptive variables chosen for classification as *classificatory variables*, and denote them by $\hat{x} = \{\hat{x}_1, \hat{x}_2, ..., \hat{x}_m\}$, where $m \leq n$. Then, classification methods relate the primitives to be classified to their corresponding classes by defining a special configurations of the classificatory variables known as hypotheses. In other words, the classifier's task is to find hypotheses that best fit the class-structure observed in the data [Gordon, 1999]. At the core of each hypothesis is a metric that evaluates the similarity between the primitives to classify and the resulting concepts; let us refer to this metric as a *similarity metric.*

Figure 3-4 illustrates the general steps undertaken in a classification task, where, Y, is the space of primitives to classify and, y_4, is a particular primitive in that space. The first step of the task consists of measuring the variables that describe y_4, represented as the set, $x = \{x_1, x_2, ..., x_n\}$, which

could represent the measures from n sensors applied on y_4. The second step consists of selecting the variables that are considered relevant, denoted by $\hat{x} = \{\hat{x}_1, \hat{x}_2, ..., \hat{x}_m\}$. In most cases this step is implicit in the first step, as it is assumed that all the sensors used for measuring the properties of y_4 are relevant. The final step takes as input the classificatory variables and relates them to the corresponding concept $c \in C$.

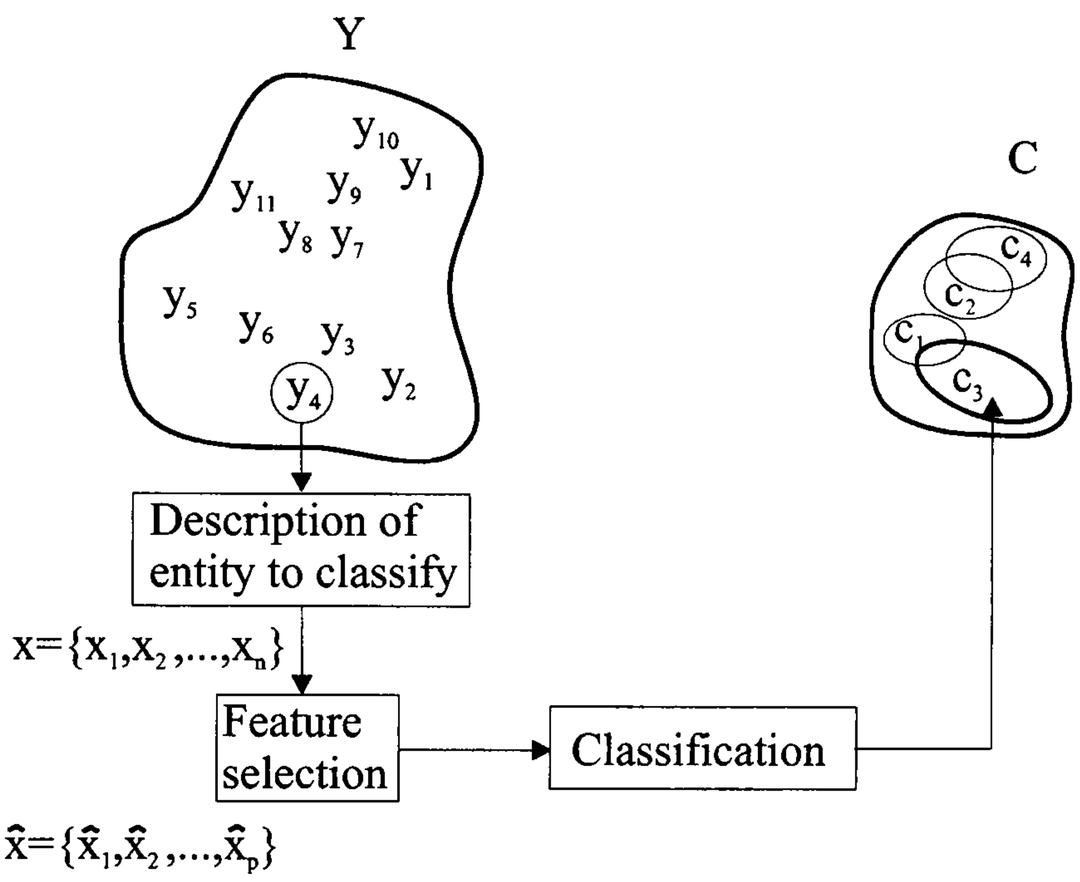


Figure 3-4: General process of classification

The following sections describe some of the existing techniques for multi-dimensional data classification, followed by a description of their limitations and the introduction of a new metric for classification based on Q-analysis, which addresses some of those limitations.

3.2.2 Distance based clustering

Clustering techniques are unsupervised classification methods which define classes or clusters of unlabelled data by exploiting some perceived regularity

or patterns of occurrence [Jain *et al.*, 1999, Gordon, 1999]. In order to find these regularities, clustering techniques commonly use classification metrics related to the geometry of the data represented in a multidimensional coordinate space.

Geometric metrics represent the primitives to classify as points in a multidimensional coordinate space. Then, a metric is defined to assess the similarity between points. A commonly used metric is the *Weighted Euclidean Distance* (WED). The WED is an instance of the Euclidean distance in which distances on the different coordinate axes are given a relevance value or weight. Then, the larger the weighted distance between primitives, the larger is their dissimilarity. WED can be calculated by applying following expression:

$$d_{ij} = \sqrt{\sum_{k=1}^{p} w_k (\hat{x}_{ik} - \hat{x}_{jk})^2}$$

where d_{ij} is the dissimilarity value between primitives i and j, $\hat{x}_{ik}$ and $\hat{x}_{jk}$ are the k-th classificatory variables of primitive i and j out of a total of p, and w_k is the weight assigned to each classificatory variable. Similarly to WED the *City block* dissimilarity metric is calculated by:

$$d_{ij} = \sum_{k=1}^{p} w_k |\hat{x}_{ik} - \hat{x}_{jk}|$$

where the variables in the expression have the same meaning as in WED.

Once a classification metric has been selected, clustering techniques classify or partition the data in such a way to reduce some error measure. For example, one could try to minimise the *Sum of Squared Error* (SSE). This is calculated as follows. Let us assume that C is a set of clusters,

$C = \{c_1, c_2, ..., c_k\}$. Let $m(c_i)$ be the mean of the point of the cluster c_i, this point is also known as the *representative of the cluster*. Assuming that $\{p_{i1}, p_{i2}, \ldots, p_{in}\}$ are the points contained in c_i; the representative of this cluster is calculated by, $m(c_i) = (\sum_{j=1}^{n} p_{ij})/n$. Then the SSE of the set of clusters C is defined as:

$$E_C = \sum_{i=1}^{k} \sum_{j=1}^{n} \|p_{ij} - m(c_i)\|^2$$

In other words, for a cluster c_i, $m(c_i)$ is the best representative of its elements, in the sense that it minimises the sum of squared error. The value of E_C depends on how the points are grouped under each cluster c. Thus an optimal clustering can be defined as the one that minimises E_C.

3.2.3 Self-organising methods

Self-organising methods are unsupervised classification techniques which define emergent classes by allowing the hypothesis to emerge through self-organisation. A good example of a self-organising classifier is the *Self Organising Map* (SOM) also known as Kohonen map [Kohonen, 1995].

Figure 3-5 illustrates the basic architecture of a SOM. This consists of M neurons usually arranged in a low-dimensional grid (two dimensional in Figure 3-5). Each neuron m_i has associated a p-dimensional weight vector $w_i = [w_{i1}, w_{i2}, ..., w_{ip}]$ with the same dimension as the input $\hat{x}$. An input $\hat{x}$ is compared to the weight vector of each of the neurons, and the neuron with the weight vector most similar to the input is considered the winning neuron (firing neuron). In a SOM the similarity between the input and the weight vector of neuron i is measured by their distance, that is: $\|\hat{x} - w_i\|$.

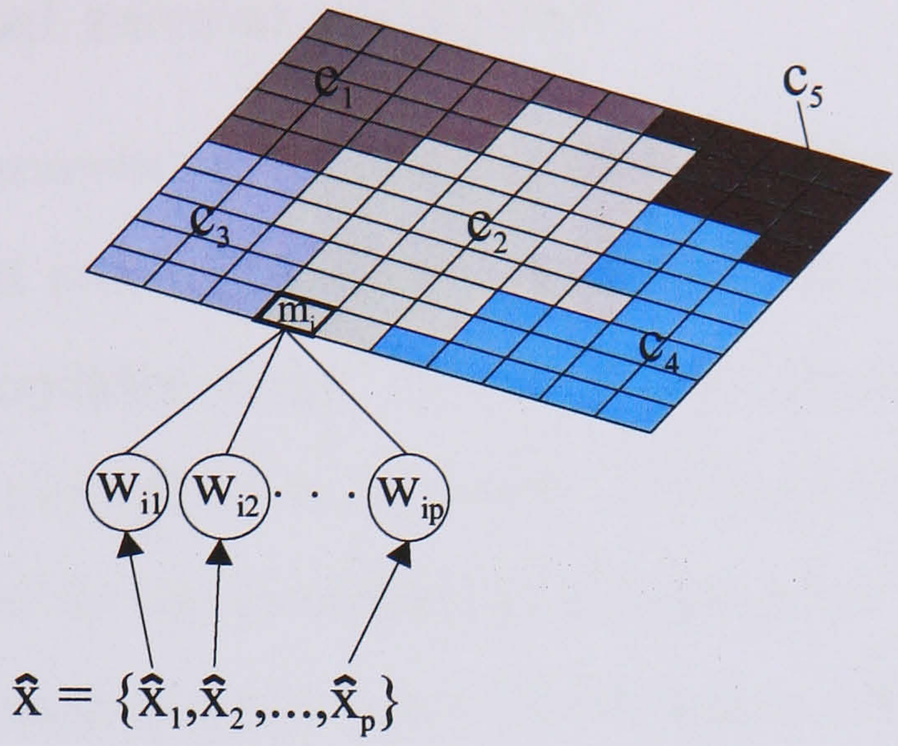


Figure 3-5: A self organising map

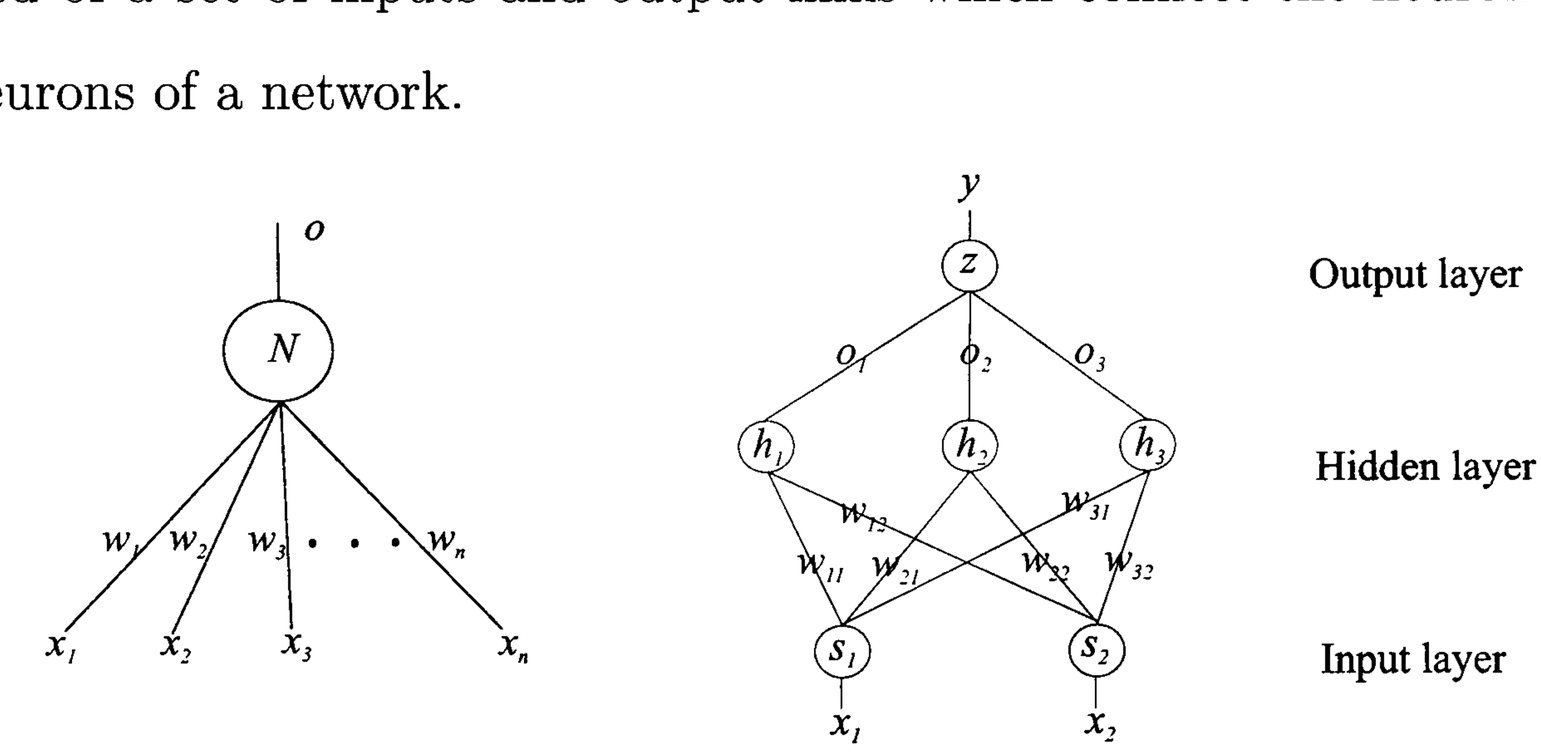
The SOM learns hypotheses competitively, i.e. the winning neuron is updated in such a way that its updated weights improve the previous matching (minimising the error between $\hat{x}$ and w_i).

Because of the spatial ordering of the neurons in the lattice, allowing the winner's neighbouring neurons to be also updated towards the input $\hat{x}$, the result is an ordered map of the input vectors observable on the surface topological map. In Figure 3-5 the surface of the map displays five areas designated by $c_1, c_2, ..., c_5$; the firing of the neurons that lie within the same area indicates that the inputs are somehow similar. Thus, classification in the SOM is the task of defining areas on the surface map and relating them with input vectors, so that when an input vector fires a neuron within an area, the input is considered to be of a class related to that area. Some examples of using the SOM for classification include [Vesanto and Alhoniemi, 2000, Wu and Chow, 2004].

3.2.4 Artificial neural networks

Artificial Neural Networks (ANN) are one of the most popular methods used in classification and pattern recognition [Bishop, 1995], mainly because of their conceptual simplicity of use, i.e. ANN learn simply by observing labelled examples of the primitives to classify. ANN are also popular because they can be applied to the classification of numeric or discrete functions, providing a broad range of applications. For example ANN could be applied to learn the motor actions of a mobile robot depending on the input sensors (numeric function) [Hesselroth *et al.*, 1994] or they could be applied to learn where to move a checkers piece given the board configuration (discrete function) [Kumar and Fogel, 1999].

The basic element used in ANN is a *neuron* or *perceptron.* A neuron is composed of a set of inputs and output links which connect the neuron to other neurons of a network.



(a) Artificial neuron

(b) Multilayer perceptron

Figure 3-6: Artificial neuron and neural network

Figure 3.6(a) illustrates a neuron N, with n inputs $\{x_1, x_2, \ldots x_n\}$, and one output link o. Each input link x_i, has a weight w_i, associated with it.

A neuron's general operation is as follows:

$$o = f(\sum_{i=0}^{n} x_i w_i)$$

where the output o is a function of the linear combination of the inputs, x_i, multiplied by their corresponding weights, w_i. The simplest function is a threshold, i.e., if the sum of weighted inputs is higher than the threshold the output is active (1), otherwise inactive (0). Another of the common functions used is the sigmoid function which is explained below.

Neurons are structured in networks, a common architecture used is the *multilayer network* or *multilayer perceptron* illustrated in Figure 3.6(b). This multilayer architecture is generally composed of three layers of neurons, an *input layer*, *hidden layer* and *output layer*, where s are input neurons, h are hidden neurons, z are output neurons. w_{ij} is the weight connecting neuron i (hidden layer) and j (input layer), o_k is the output of the k-th hidden neuron, x are inputs and y the output.

Each element of the input vector is introduced to each of the neurons in the input layer. Neurons of the input layer are simply used to connect the input to the neurons of the hidden layer. Neurons at the hidden layer apply the previously described function. For example, sigmoid units output a continuous value between 0 and 1. In particular, a sigmoid unit performs the function:

$$o_{h_i} = \frac{1}{(1 + e^{\sum_{j=0}^{n} w_{ij} x_j})}$$

where o_{h_i} is the output value of neuron h_i, w_{ij} is the weight value connecting the neuron j of the input layer and neuron i of the hidden layer, x_j is the

value of the input neuron j, and n is the total number of input neurons. As can be seen, the sigmoidal function depends on, $\sum_{j=0}^{n} w_{ij}x_j$, which is the linear combination of inputs and weights of the neuron.

ANN learn and represent hypotheses by adjusting the weights of the neurons in a network. When used in a supervised manner, ANN are presented with labelled training examples and use algorithms such as *backpropagation* to update their weights.

3.2.5 Incremental concept formation methods

Concept formation methods originate from ideas introduced by *conceptual clustering* [Michalski, 1980, Michalski and Stepp, 1983]. In conceptual clustering, the similarity between the primitives to cluster is not only based on their individual properties (e.g. their distance), but also takes into account a set of predefined concepts that describe the classes.

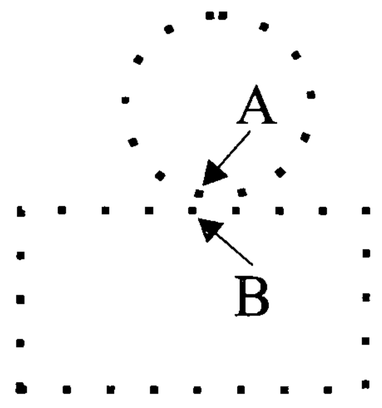


Figure 3-7: Example of conceptual clustering

Figure 3-7 illustrates two points, A and B, which would be clustered together by taking into account only their distance as a dissimilarity measure. However if the concepts of *square* and *circle* are known to the classifier, then the points would be clustered with their corresponding concepts rather than together. In conceptual clustering, the primitives to cluster are represented

by finite and discrete variables, and concepts are predefined in variable valued logic calculus [Michalski, 1980].

Incremental concept formation methods are unsupervised classification methods that similarly to conceptual clustering, classify observed primitives by taking into account a set of concepts [Gennari *et al.*, 1989, Fisher, 1987]. The differences from concept clustering are that (i) concepts are not predefined, but acquired incrementally, and (ii) concepts are ordered by generality in a hierarchy. The hierarchy of concepts is known as a *concept hierarchy*, and is used to encode the system's knowledge.

Concept hierarchies represent concepts in a tree-like structure, where nodes contain concepts represented as attribute-value pairs. The links of the tree are used to perform tests on the attributes of the primitives to classify. Starting from the top concept, if the attributes of the primitives correspond to those associated with one of the links, then the primitive is sorted through that link. This process is repeated until the primitive reaches an end concept, at which point the primitive is classified as part of this end concept.

An important aspect of concept formation methods is that the concept hierarchy is not provided *a priori*, but created incrementally and dynamically as the system classifies new primitives. Examples of incremental concept formation methods include CLASSIT [Gennari *et al.*, 1989], COBWEB [Fisher, 1987] and UNIMEN [Lebowitz, 1987].

3.3 Limitations in multidimensional data classification

The previous section introduced multidimensional spaces and presented some methods for defining classes. This section provides a review of those methods by indicating their strength and weaknesses. This review focuses on the following characteristics of multidimensional classification methods: *similarity metrics*, *feature selection problem* and *interpretability of hypothesis.*

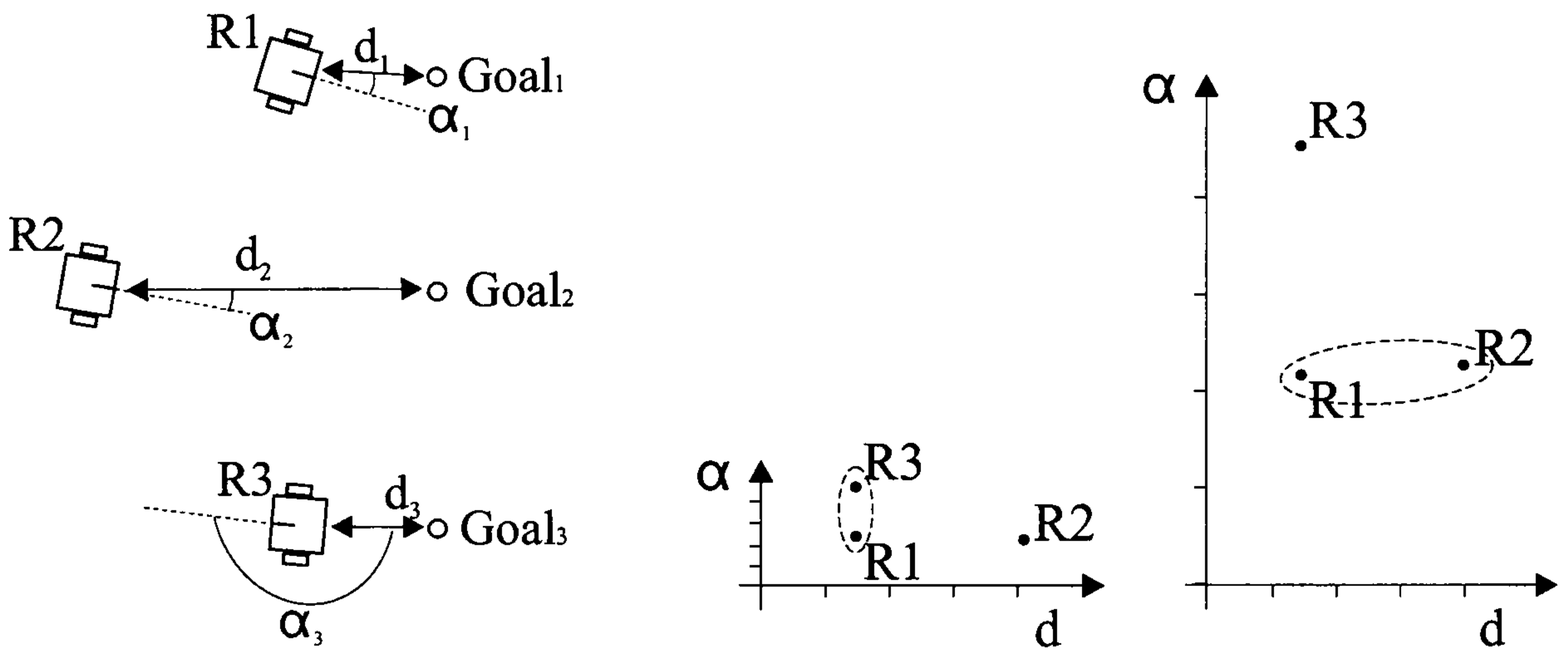
3.3.1 Similarity metrics

All of the previous classification methods assume a metric to evaluate the similarity or dissimilarity between data points, known as a similarity metric.

One of the commonly used similarity metrics is based on representing data-points in a multidimensional coordinate space, and measuring some of their geometric properties such as the Euclidean distance between them. This means that the information of a set of classificatory variables is subsumed in a set of distances, thus it is of critical importance that these distances reflect accurately the relevant relations in the data [Gordon, 1999, Duda and Hart, 1973].

To exemplify this criticality, let us consider the following example. Figure 3.8(a) illustrates three robots (R1, R2, R3) in different configurations with respect to a goal. Each configuration is represented by two different variables, namely, a distance to goal d, and an angle to goal α. Each configuration can be represented as a point in a two-dimensional coordinate space, as illustrated in Figure 3.8(b) where two possible representations are given. In the first

coordinate space, the d variable is more relevant, as a change of one unit in d incurs in a larger Euclidean distance between points in relation to a unit change of α. In this coordinate space, robots R1 and R3 are in more similar situations than R2. In the second coordinate space, the variable angle to goal (α) has been scaled differently, i.e. 'stretched' or given more relevance than d. As an effect of this, R1 and R2 are now in a more similar configuration than R3.



(a) Mobile robots (b) Similarity based on Euclidean distance

Figure 3-8: Mobile robots in different configurations and their similarity based on Euclidean distance

This type of coordinate space is known as *isotropic* that is, irrelevant to translations and rotations but not to linear transformations such as simple scalings [Duda and Hart, 1973]. This example shows the importance of finding a distance value that accurately represents similarity among the particular configurations. This issue is sometimes known as the *chalk and cheese problem* [Johnson and Picton, 1990]*:* what is the meaning of measuring the chalkiness in relation to measuring the cheesiness of an object? In other words, how can different variables be compared?

As seen previously, clustering techniques (Section 3.2.2) and SOMs (Section 3.2.3) exploit Euclidean distance as their similarity metric. Clustering techniques use this distance metric explicitly by defining classes or clusters based on it. SOMs use this distance metric implicitly by using it to find which of the neurons is most similar to the input. Thus both methods are prone to the issue of defining a correct distance metric.

3.3.2 Feature selection problem

As seen in Section 3.1.3 multidimensional spaces are commonly used to describe the primitives to classify. For example in Figure 3-8 the variables distance d, and angle α, construct a two-dimensional space used to represent the position of a robot relative to a goal. In a more complex environment such as robot football, one could use many more variables to describe the state, including the following: relative position of all players, speed of players, relative positions of all landmarks, position of the ball, speed of the ball, current player with ball, score, time left, etc. In this complex situation, choosing the relevant set of variables for a particular task (e.g. decision making) is not trivial. Usually, this selection depends on the expertise of the system's designer.

In the previous context, it would be tempting to have a robot with extensive sensing capabilities (including all of the variables a designer could think of) and searching for different combinations of relevance; but this approach soon becomes impractical, as adding dimensions or variables in the representation of state incurs on the curse of dimensionality problem (see Section 2.4.1). The issue then becomes whether: (i) to add only the relevant

variables to the description or (ii) to add all variables and filter the irrelevant ones. In most practical cases, the first option is chosen, in which the designer provides a description of the state which is relevant to the task. Once an appropriate description is chosen, generalisation methods can be applied to the relevant variables if these are still too large (see Section 2.4.2).

Automating this process of discovering relevant variables or features is known in the literature as *feature selection.* The main task of feature selection is: given a set of n features describing some entities, selecting a subset of m relevant features, where $m < n$, such that, the subset provides the same or similar information about the entities [Dash and Liu, 1997, John *et al.*, 1994]. Usually, the relevance of a feature is measured in the context of classification, where relevant features are those which provide useful information for discriminating between entities of different classes.

Statistical techniques exist for reducing dimensions in high dimensional spaces. For example, *principal component analysis* (PCA) [Kendall, 1980] is a method by which high dimensional spaces are reduced and represented by their *principal components* or simply, *components.* Principal components are new dimensions defined as linear combinations of the original dimensions. In other words, the descriptive variables are linearly combined into new variables. Principal components are defined by the decreasing variance of the data they represent, i.e. the first component represents the dimension in which the data has the highest variance, the second component represents the dimension with the second highest variance, and so on. An important characteristic of components is that they are orthogonal to each other. The final aim of PCA is to represent a large percentage of the data total variance

with few components (2 or 3). Although this method reduces the number of dimensions, the resulting components or relevant dimensions are still linear combinations of possibly irrelevant dimensions. This implies that in searching for the components the method must also search along the irrelevant dimensions. Moreover, as PCA is based on the data total variance, and this depends on the variance of each of the dimensions, it will also be necessary that each dimension is appropriately normalised.

None of the previous classification methods has the capability of filtering irrelevant dimensions or variables of their input representation, i.e., they generate classes based on the combination of *all* the given variables, even if these are irrelevant in relation to the classes.

3.3.3 Interpretability of hypothesis

Having systems that are easy to inspect facilitates their analysis and the discovery of errors. For example, if a classifier is consistently misclassifying some inputs, then being able to inspect the system and understand the reason for such an anomaly is a desirable characteristic.

As seen above, different classification methods represent hypotheses in different ways. For example, clustering techniques represent hypotheses as a set of clusters, $C = \{c_1, c_2, ..., c_n\}$. A particular primitive is then either part of the cluster (concept) or not, according to the similarity metric used. How could one interpret the meaning of these clusters or concepts? In low dimensional spaces (up to three dimensions) one could say that the concept is related to different areas of the multidimensional space, by looking at a plot of their geometrical location. Higher order dimensions would require

other visualisation methods.

ANNs represent hypotheses implicitly in the weights and thresholds associated with each neuron of the network. From this viewpoint, ANNs are *black boxes* which classify primitives but don't allow the inspection of the hypothesis used to do so.

Concept formation methods represent concepts and the primitives to be classified using the conjunction of attribute-value pairs. For example, the concept of a 'bird' could be expressed by the following attribute-value pairs: $(nature = animal) \wedge (locomotion = wings) \wedge (size = small)$. This type of representation can be inspected with relative ease, and is similar to the hypothesis used by the method presented in the following section.

3.4 Q-analysis and relational concepts

The previous sections of this chapter have introduced concepts as generalisations of primitives described by multidimensional data; they have also reviewed some of the existing methods used in classification and concept learning, discussing their strengths and limitations.

This section presents the necessary theory for the development of a new classification method based on the well established methodology of Q-analysis [Atkin, 1977, Atkin, 1981]. This novel method addresses some of the limitations identified in current classification methods. That is, (i) given the limitations of geometric models based on distance similarity metrics, the methodology of Q-analysis offers new insights to the concept of similarity. (ii) Q-analysis allows the possibility of finding relevant or irrelevant dimensions or variables, thus addressing the feature selection problem and the curse

of dimensionality. (iii) The Q-analysis methodology uses relatively simple representations of multidimensional data and its operators. This results in hypotheses that are easily interpretable, i.e. easy to analyse without any *black boxes.*

As Q-analysis methodology considers the relational structure and multidimensional connectivity of a set of elements, we refer to concepts or classes generated following this methodology as *relational concepts*.

3.4.1 Introduction to Q-analysis

Q-analysis is a multidimensional generalisation of network theory introduced by Atkin [Atkin, 1977, Atkin, 1981], which is able to model general *n-ary* relations between the variables describing some primitives or elements of a multidimensional set. This analysis is especially suited for discovering *relational structures* in multidimensional data.

In Q-analysis, the similarity between primitives is no longer defined as a distance, but is based on structural ideas of connectivity between the primitives. This is in marked contrast to methods of classification that map objects into multidimensional data spaces, and cluster them into components based on distance similarity metrics. Through its notion of 'q-connection' it provides a graded method of classification according to shared dimensions or variables.

As an example, consider a robot with the characteristics: $robot_1 = \langle biped,$ $battery, camera, PC\rangle$ and another with the characteristics: $robot_2 = \langle wheeled,$ $solar, camera, PC\rangle$. These robots are similar through sharing the characteristics $\langle camera, PC\rangle$. Each characteristic can be considered to be a vertex in

a multidimensional space. Then, $robot_1$ can be represented by a tetrahedron (3-dimensional polyhedron) with vertices: *biped*, *battery*, *camera* and PC. Similarly, $robot_2$ can be represented by another tetrahedron with vertices: *wheeled*, *solar*, *camera* and PC. These polyhedra are illustrated in Figure 3-9, which also shows the shared characteristics with vertices: *camera* and PC. In this case the shared characteristics form a line (1-dimensional) and the tetrahedra are said to be 1-connected. In general the more highly connected these polyhedra are through their shared characteristics, the more similar they are. Thus the polyhedra can be clustered into classes according to their similarity as measured by their connectivity.

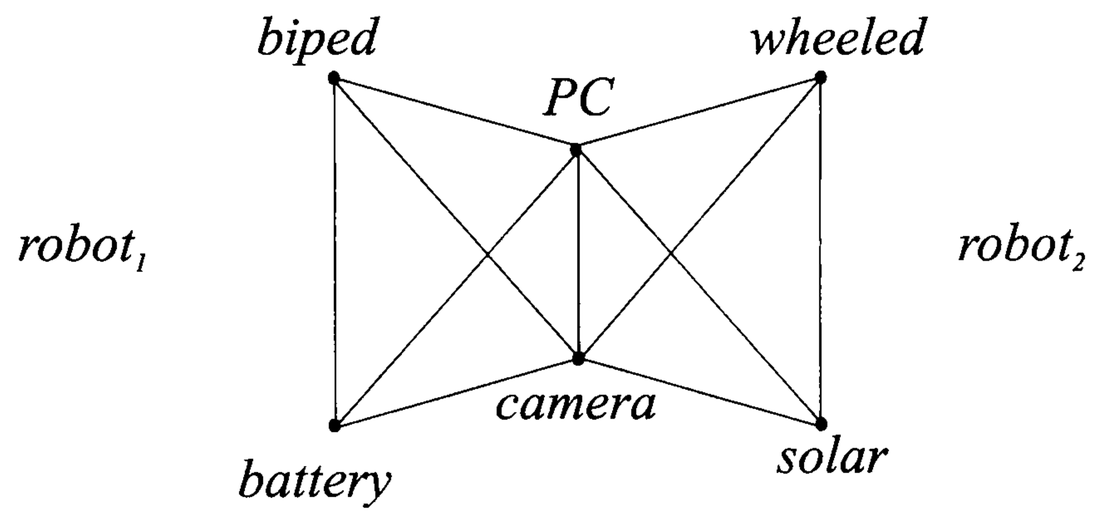


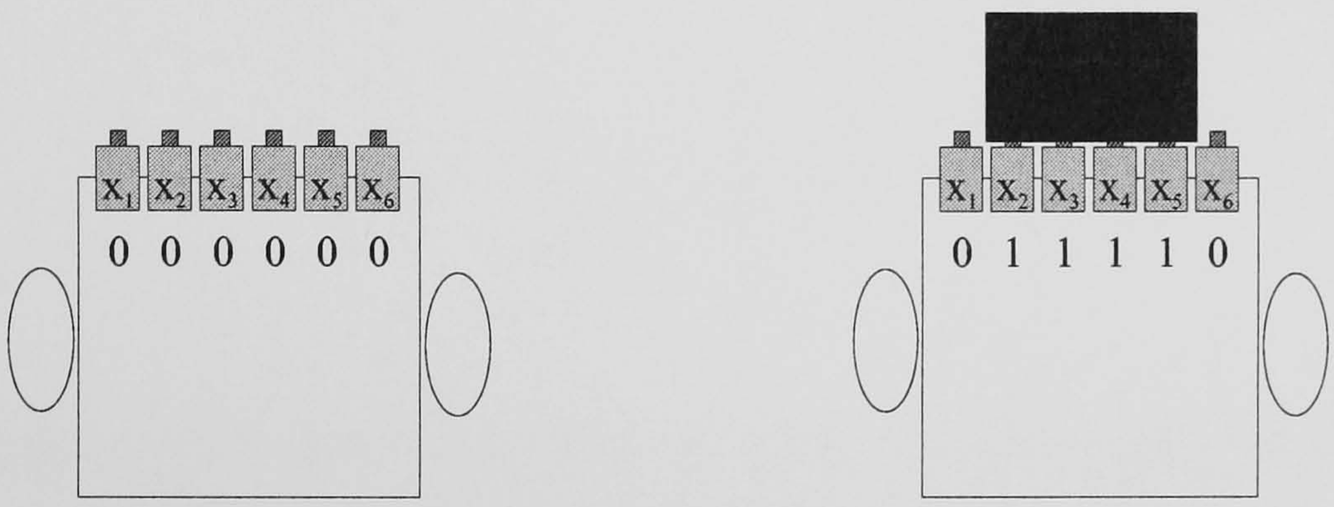
Figure 3-9: Tetrahedra representing the characteristics of two robots

As will be explained, the concepts or classes resulting from a Q-analysis classification are called *relational concepts*, as these are based on the *n-ary* relations between the variables describing the primitives.

The following sections describe the basic notions and techniques used in the Q-analysis methodology.

3.4.2 An incidence matrix representation

Relational data can be represented using an *incidence matrix*, as follows. Let us assume that a robot's sensory input is composed of six binary touch sensors $\{x_1, x_2, \ldots, x_6\}$, as illustrated in Figure 3-10. The state of the robot at any time will be defined as the combination of the states of the individual sensors, x_i, and denoted by s. Figures 3.10(a) to 3.10(d) illustrate a robot sensing four different states.



(a) Sensing s_1 (b) Sensing s_2

(c) Sensing s_3 (d) Sensing s_4

Figure 3-10: A robot sensing different environmental states

If the off/on states of the sensors are represented as 0/1 values, each of the previous states can be represented as a row in the matrix, M (Table 3.1). This is called an incidence matrix because it shows the incidence of the relationship between the states (rows) and the sensors (columns). In the

incidence matrix a value of 1 in row i and column j indicates that sensor j is *activated* in state i. The value 0 means that the sensor is not activated for that state.

Table 3.1: Incidence matrix M

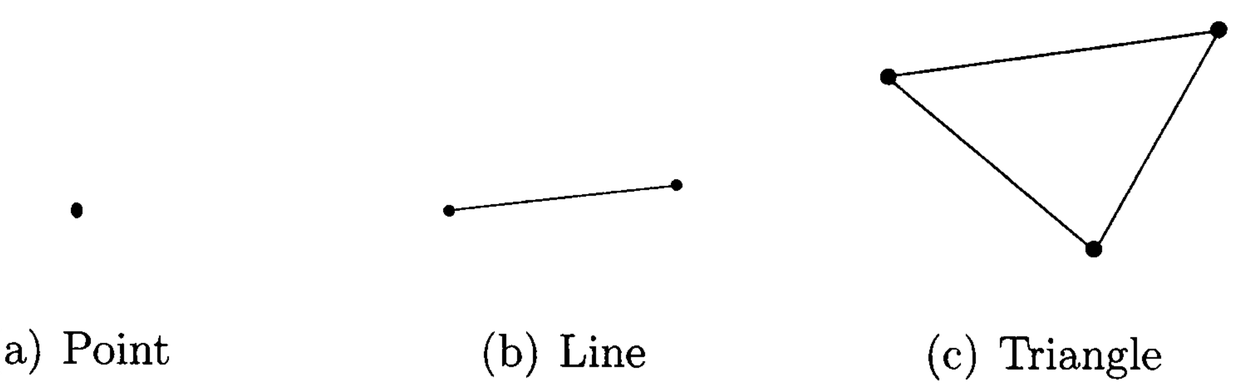
	x_1	x_2	x_3	x_4	x_5	x_6
s_1	0	0	0	0	0	0
s_2	0	1	1	1	1	0
s_3	0	1	1	1	0	0
s_4	1	1	0	0	1	1

Although typical robotic sensors provide their responses in continuous ranges (e.g. a luminosity sensor may provide a response from 0 to 255 proportional to the luminosity it receives), the assumption of binary sensors can be generalised to continuous variables, as will be shown in the following chapters.

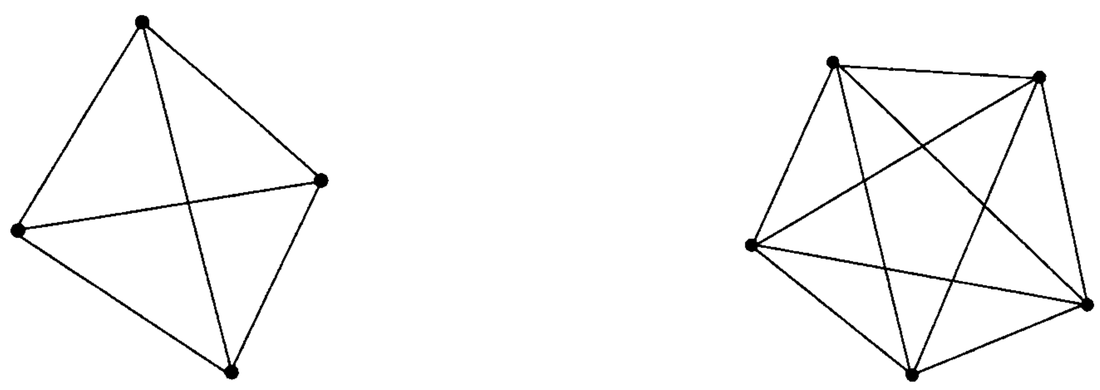
3.4.3 Simplex

In general, a relation between two sets can determine a new object called *simplex*. For example let S be a set of states and X a set of sensors. Then s_i is related to x_j if x_j is activated for that state (as seen in the previous section). Let s_i be related to the active sensors $x_0, x_1, ..., x_n$, then the object denoted by $\langle x_0, x_1, ..., x_n \rangle$ is called an n-dimensional simplex. This terminology comes from algebraic topology, in which an n-dimensional polyhedron has $(n+1)$ vertices [Johnson, 1981].

For example, a simplex with one vertex is a 0-dimensional point (Figure 3.11(a)), a simplex with two vertices is a 1-dimensional line (Figure 3.11(b)), a simplex with three vertices is 2-dimensional triangle (Figure 3.11(c)), a simplex with four vertices is a 3-dimensional tetrahedron (Figure 3.11(d)), a simplex with five vertices is 4-dimensional a 5-hedron (Figure 3.11(e)), and so on.



(a) Point (b) Line (c) Triangle

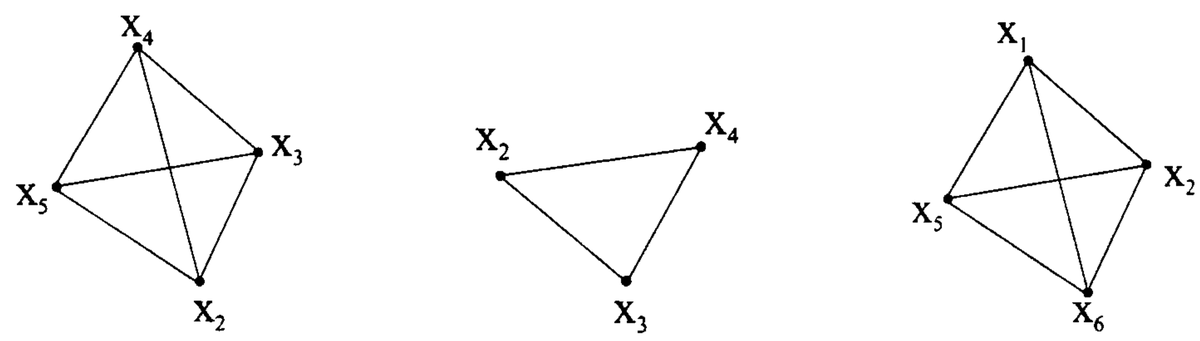


(d) Tetrahedron (e) 5-hedron

Figure 3-11: Example of simplices

Each row of an incidence matrix can determine a simplex, with vertices corresponding to those columns to which it is related. For example, let $\sigma(s_4)$ be the simplex associated with state s_4 in Figure 3.10(d). Then we can write $\sigma(s_4) = \langle x_1, x_2, x_5, x_6 \rangle$, which is a tetrahedron. Similarly, $\sigma(s_2) = \langle x_2, x_3, x_4, x_5 \rangle$ and $\sigma(s_3) = \langle x_2, x_3, x_4 \rangle$. In this example, state s_1 is associated with no sensors being activated. Thus s_1 is associated with the *null simplex*, denoted σ_{-1}, which has no vertices. In summary, this notation allows us to represent the state observed by a robot as a simplex or polyhedra in

multidimensional spaces. Figure 3-12 illustrates the simplices and polyhedra related to the states observed by the robot in Figure 3-10.



(a) $\sigma(s_1)$ (b) $\sigma(s_2)$ (c) $\sigma(s_3)$ (d) $\sigma(s_4)$

Figure 3-12: Simplices representing the robot's sensory state

3.4.4 Hierarchical decomposition of simplices

An important idea in Q-analysis is that high dimensional simplices can be decomposed into their lower order simplices called their *faces*. For example, the simplex representing $\sigma(s_2) = \langle x_2, x_3, x_4, x_5 \rangle$ is a tetrahedron which is composed of the following faces: four triangles, six lines, and four vertices as shown in Table 3.2.

Table 3.2: Sub-simplex hierarchy

q-dimension	face simplices
3	$\sigma(s_2) = \langle x_2, x_3, x_4, x_5 \rangle$
2	$\langle x_2, x_3, x_4 \rangle$ $\langle x_2, x_3, x_5 \rangle$ $\langle x_2, x_4, x_5 \rangle$ $\langle x_3, x_4, x_5 \rangle$
1	$\langle x_2, x_3 \rangle$ $\langle x_2, x_4 \rangle$ $\langle x_2, x_5 \rangle$ $\langle x_3, x_4 \rangle$ $\langle x_3, x_5 \rangle$ $\langle x_4, x_5 \rangle$
0	$\langle x_2 \rangle$ $\langle x_3 \rangle$ $\langle x_4 \rangle$ $\langle x_5 \rangle$

As will be shown in the following sections, this idea of hierarchical decomposition allows one to study the relations among simplices, and thus the multidimensional data they represent, at different q-dimensional levels.

This means, that simplices can be considered connected at different levels of connectivity.

3.4.5 q-nearness, q-connectivity and structural similarity

Let the intersection of two simplices be defined as their largest shared face. For example, $\sigma(s_2) \cap \sigma(s_4) = \langle x_2, x_3, x_4, x_5 \rangle \cap \langle x_1, x_2, x_5, x_6 \rangle = \langle x_2, x_5 \rangle$. The shared face is illustrated in bold in Figure 3-13. As the shared face of these simplices has a dimension of 1 (line), $\sigma(s_2)$ and $\sigma(s_4)$ are said to be *1-near*.

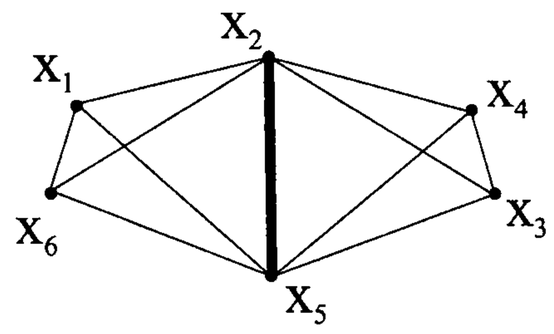
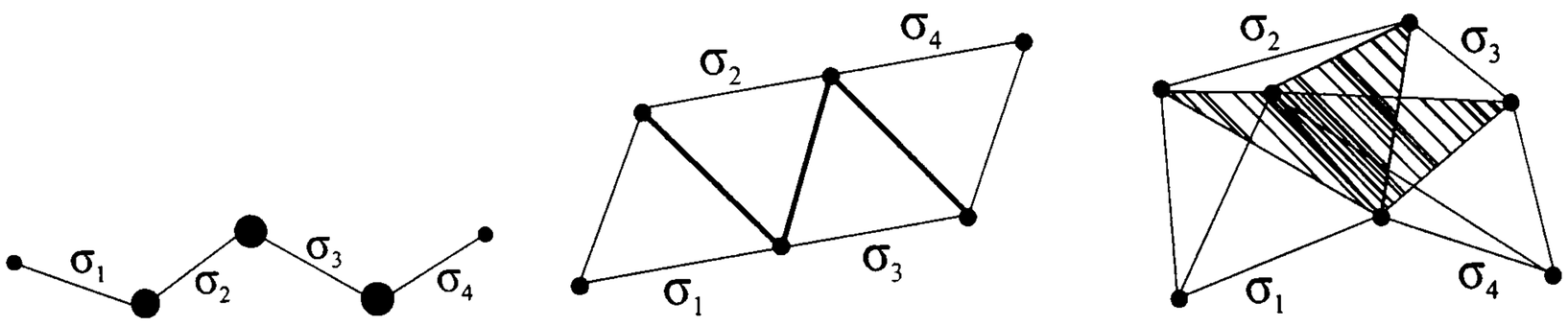


Figure 3-13: q-nearness of two simplices

Two simplices σ and σ' are said to be *q-connected* if there is a chain of pairwise p-near simplices between them, such that, $p \geq q$. For example, Figure 3.14(a) illustrates a chain of four 1-simplices (lines), each of them 0-near (through a point) to their neighbouring simplex, thus the set of four 1-simplices are 0-connected. Figure 3.14(b) illustrates a chain of four 2-simplices (triangles), each of them 1-near (through a line). Figure 3.14(c) illustrates a chain of four 3-simplices (tetrahedrons), each of them 2-near (through a triangle).



(a) 0-connected (b) 1-connected (c) 2-connected

Figure 3-14: Example of chains of q-connected simplices

When the simplices are represented on an incidence matrix, such as, M, their q-nearness can be simply calculated as, $MM^T - \mathbf{1}$, where, M^T, is the transpose of M and $\mathbf{1}$ is a matrix with all elements equal to 1. The result of this operation on the incidence matrix in Table 3.1 is illustrated in Table 3.3 and is known as a *shared face matrix* .

Table 3.3: Shared face matrix corresponding to M in Table 3.1

$MM^T - 1 =$

	s_1	s_2	s_3	s_4
s_1	-1	-1	-1	-1
s_2	-1	3	2	1
s_3	-1	2	2	0
s_4	-1	1	0	3

The shared face matrix represents the direct connectivity of the simplices based on their shared vertices. Every p-simplex is p-near to itself, and this is shown in the diagonal of the matrix, e.g. $\sigma(s_2)$ and $\sigma(s_4)$ are 3-simplices, $\sigma(s_3)$ is a 2-simplex and $\sigma(s_1)$ is a null simplex. Looking out from the diagonal, the shared face matrix represents the connectivity between pairs of

simplices. For instance, in this case $\sigma(s_2)$ is 2-near to $\sigma(s_3)$, $\sigma(s_4)$ and $\sigma(s_2)$ are 1-near, $\sigma(s_3)$ and $\sigma(s_4)$ are 0-near and $\sigma(s_1)$ is not connected to any of the other rows of the incidence matrix, M.

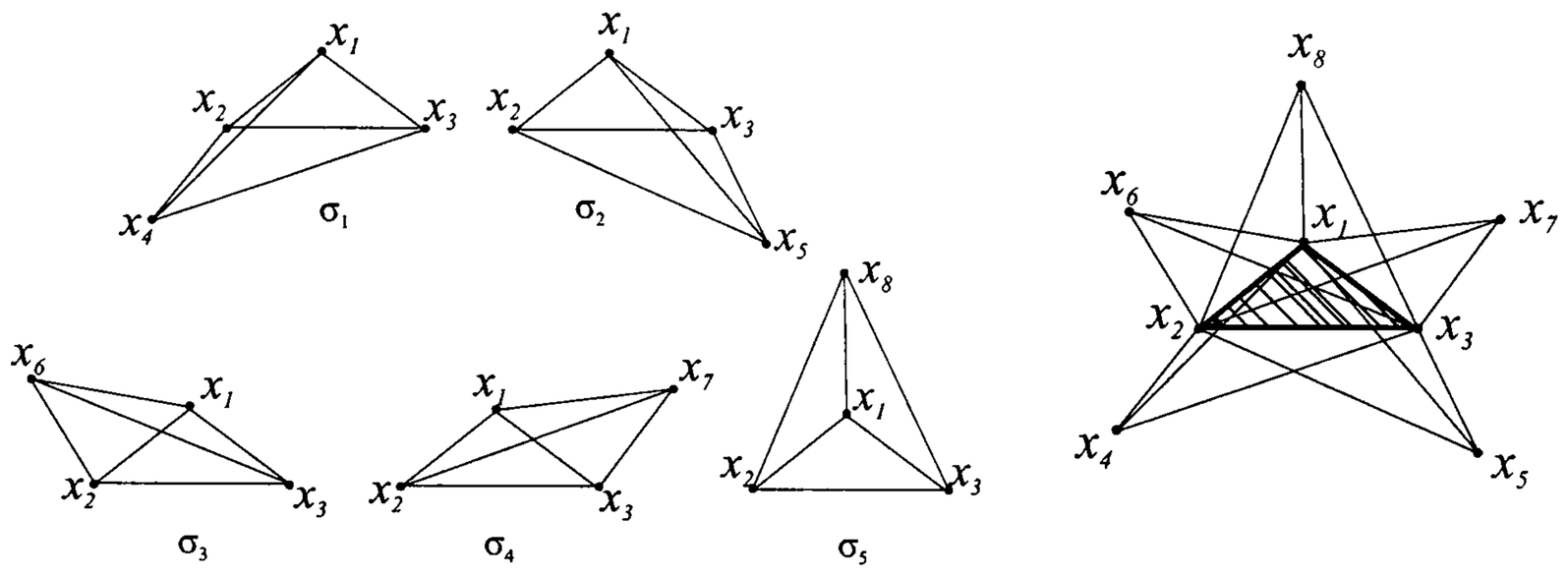
The q-nearness of two simplices is a fundamental measure of their structural similarity, that is, the higher the dimension of their shared face, the higher their structural similarity. In the extreme case, when two simplices share all their faces, they are identical with respect to their descriptive dimensions.

As described earlier, the aim of a classification task is to relate a set of primitives to their corresponding classes. A similarity measure based on q-nearness provides the relation of a pair of simplices, thus an important issue is to extend the idea of q-nearness to a similarity measure applicable to a set of simplices. One could think of q-connectivity as a measure of similarity between a set of q-near simplices, but due to its pairwise definition, this measure is not well suited for classification. For example, any of the simplices illustrated in Figure 3-14 have the same q-connectivity value, but taking σ_1 and σ_4 of any of the chains, illustrates how these simplices do not share any vertex, but are still q-connected. This is an effect of q-nearness being pairwise transmitted along the chain. Thus, using q-connectivity as a classification metric for a set of simplices would consider as similar two possibly unconnected simplices.

To extend the previous idea of q-nearness as structural similarity to a set of simplices, and avoiding the problem of pairwise transmission of q-connectivity, the following section introduces another element of the Q-analysis methodology known as a *hub*.

3.4.6 Hubs and stars

Given any set of simplices, $\sigma_1, \sigma_2, \ldots, \sigma_n$, their *hub* is the largest shared face of them all [Johnson, 1986]. Thus, $\mathbf{hub}(\sigma_1, \sigma_2, \ldots, \sigma_n) = \cap_{i=1}^{n} \sigma_i$. Figure 3.15(a) illustrates five 3-simplices: $\sigma_1 = \langle x_1, x_2, x_3, x_4 \rangle$, $\sigma_2 = \langle x_1, x_2, x_3, x_5 \rangle$, $\sigma_3 = \langle x_1, x_2, x_3, x_6 \rangle$, $\sigma_4 = \langle x_1, x_2, x_3, x_7 \rangle$ and $\sigma_5 = \langle x_1, x_2, x_3, x_8 \rangle$. The hub of these simplices is: $\langle x_1, x_2, x_3 \rangle = \mathbf{hub}(\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5)$, illustrated in Figure 3.15(b) as a greyed triangle. All of the possible simplices that share a same hub are known as the hub's *star* [Johnson, 1986] .



(a) Simplices $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ and σ_5 (b) hub$(\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5)$

Figure 3-15: Some simplices and their hub

Hubs and stars allow us to define a novel method for the classification of instances, based on the idea of hubs being used as the defining structure or core of a class, i.e., the necessary and sufficient conditions for primitives to belong to a class. In this thesis, hubs used for classification are known as *classifier hubs*. The classification is based on the idea of representing classes as hubs, and primitives belonging to a class as the stars of a hub. This novel method for classification is further explained in the following sections.

3.4.7 Classification using classifier hubs

The idea behind classification using *classifier hubs* is that a set of simplices sharing a hub, i.e. the hub's star, can be considered similar, thus being part of the same class, if their hub is relevant for the given class. Relevant hubs are here known as classifier hubs. For example, a class *bird* described by the simplex: $\langle alive, wings, non\text{-}mammal, vetebrate, multicolour, small, fisher, hunter, nocturnal, feathers\rangle$ could have the following classifier hub: $\langle alive, wings, non\text{-}mammal, vertebrate\rangle$, thus any animal sharing this essential characteristics would be classified as a *bird*, irrespective of the value of the other variables. This last remark is important in the sense that the faces which are not contained in the classifier hub are in some sense irrelevant for that particular class. Classifier hubs represent the core structure of a concept, and thus are considered the representatives of the concept.

The classification task can now be seen as the task of finding a set of classifier hubs, whose combination results in the desired hypothesis. In our terms, classifier hubs are referred to as *relational concepts* as they provide the relational structure of the concept or class they represent.

The following sections exemplify how to use the Q-analysis methodology for classification. The examples are based on supervised classification using two different data-sets. These examples illustrate some heuristics to identify classifier hubs, and demonstrate how classifier hubs filter irrelevant variables or features.

3.5 Finding classifier hubs

In this example, the CorrAL data-set has been used [Dash and Liu, 1997]. This is a synthetic data-set that contains one target concept ($C = 1$), and uses six binary variables to describe each primitive, namely ($A0$, $A1$, $B0$, $B1$, I, K). A primitive is defined as belonging to the target concept (C=1) if the following expression is evaluated as true: $(A0 \wedge A1) \vee (B0 \wedge B1)$. Thus ($A0$, $A1$, $B0$, $B1$) are relevant variables with respect to the concept; I, is an irrelevant variable, and K is a variable correlated to the target concept 75% of the time. In total, the data-set contains 40 primitives, 20 of which belong to the target concept. This data-set is illustrated in Table 3.4, where the first 20 primitives do not belong to the target concept ($C = 0$).

The method to find the relational concepts consists of the following steps:

1. Find the hubs in the data using the *star-hub analysis*.

2. From the total set of hubs encountered, select a sub-set of classifier hubs.

The first step in the method identifies *all* of the hubs that exist in the data. The second step selects a sub-set of relevant hubs and defines them as relational concepts (classifier hubs). These two steps are explained in more detail in the following.

3.5.1 Star-hub analysis

The star-hub analysis takes each of the data primitives, represented as simplices, and searches for their shared hubs. Broadly speaking, this is done by intersecting the simplices and finding their shared faces. The star-hub

analysis used in this thesis associates each hub with two statistical measurements that indicate the number of simplices of each class sharing the same hub. These statistics are named *specificity* and *broadness* and are further explained below. Table 3.4 illustrates the data-set and in bold is represented the hub most shared in the data-set, i.e. $\langle A1 \rangle$.

Table 3.4: CorrAL data-set

*A*0	*A*1	*B*0	*B*1	*I*	*K*	*C*
0	0	0	0	0	0	0
0	0	0	1	1	0	0
0	0	1	0	0	1	0
0	1	0	0	0	0	0
0	**1**	0	1	1	0	0
0	**1**	1	0	1	1	0
1	0	0	0	0	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
0	**1**	1	0	1	0	0
0	0	0	0	0	0	0
0	0	0	1	1	1	0
0	0	1	0	0	0	0
0	**1**	0	0	0	0	0
0	**1**	0	1	1	1	0
0	**1**	1	0	1	0	0
1	0	0	0	0	0	0
1	0	0	1	1	0	0
1	0	1	0	0	0	0
0	**1**	1	0	1	0	0

A0	A1	B0	B1	I	K	C
0	0	1	1	0	1	1
0	**1**	1	1	0	1	1
1	0	1	1	1	1	1
1	**1**	0	0	0	0	1
1	**1**	0	1	0	1	1
1	**1**	1	0	1	1	1
1	**1**	1	1	0	0	1
1	**1**	0	1	0	1	1
0	1	1	0	1	1	1
0	**1**	1	1	0	1	1
0	0	1	1	1	0	1
0	**1**	1	1	0	1	1
1	0	1	1	0	1	1
1	**1**	0	0	1	1	1
1	**1**	0	1	0	0	1
1	**1**	1	0	1	1	1
1	**1**	1	1	0	0	1
1	**1**	0	1	0	1	1
0	**1**	1	1	0	1	1
0	**1**	1	1	0	1	1

Table 3.5 illustrates a selection of hubs from the CorrAL data-set, ordered by their probability of occurrence or broadness. For example, hub $\langle A1 \rangle$ is the hub with highest probability of occurrence: it contains 24 simplices out of

the total 40. The table also illustrates the two statistic measures, specificity and broadness. The specificity of a hub is the maximum class conditional probability given a hub, i.e., the maximum probability of any simplex being of class C when it shares hub H, that is $P(C|H)$. For example, the probability of a simplex being of the target class, given the hub $\langle A1 \rangle$, is 16/24, because the hub is shared by 24 hubs from which 16 are of the target class.

Table 3.5: A selection of hubs from the CorrAL data-set

hub	#target	#¬target	total	specificity	broadness
$\langle A1 \rangle$	16	8	24	16/24	24/40
$\langle C \rangle$	15	5	20	15/20	20/40
$\langle A0 \rangle$	12	6	18	12/18	18/40
$\langle I \rangle$	7	10	17	10/17	17/40
$\langle A0, A1 \rangle$	10	0	10	10/10	10/40
$\langle B0, B1 \rangle$	10	0	10	10/10	10/40
$\langle A1, B0, C \rangle$	8	1	9	8/9	9/40
$\langle A0, A1, I, C \rangle$	3	0	3	3/3	3/40

There are three important observations we can make from Table 3.5. (i) In general, hubs of higher q-dimension are shared by fewer simplices than hubs with lower dimension. As high dimension hubs pose more requirements (number of vertices) to be satisfied, fewer simplices satisfy them. (ii) Some hubs contain only simplices related to a unique class (specificity = 1). For example, hub $\langle A0, A1 \rangle$ is shared by 10 simplices, and all of them are of the target class. In principle, such hubs are good for classification, as a simplex sharing this hub has a high probability of belonging to the target class. (iii) Hubs with few vertices and low specificity, e.g. $\langle A1 \rangle$ has a specificity of $16/24 \approx 0.66$; when taken in combination with other vertices can become more specific, e.g., $A1$, taken in combination with, $A0$, $\langle A0, A1 \rangle$,

has a specificity of 1. This means that a vertex could be a 'weak' classifier (low specificity) when taken individually, but become a 'strong' classifier in combination with other vertices.

3.5.2 Heuristic selection of classifier hubs

The total number of hubs found in a data-set is potentially very large, e.g. the small CorrAL data-set contains approximately 60 different hubs. Moreover, not all of these hubs are interesting for classification, as many represent the connection of different classes through irrelevant or noisy variables, for example, the hub composed of the irrelevant variable $\langle I \rangle$ is shared by the 7 target and 10 no-target concepts. This means that 7 target simplices become connected to 10 no-target simplices through the irrelevant vertex.

Ideally, a small set of classifier hubs must be selected from the total set of hubs. To this end, the two previously introduced statistical measures are used. The heuristic method to select classifier hubs operates as follows: (i) hubs are ordered by their broadness, that is, starting from the hubs with higher probability of occurrence to the ones with lower probability; (ii) starting from the broadest hub, a hub is selected as a classifier if its specificity is higher than a threshold, and if it is shared by a minimum number of simplices. The threshold and the minimum number of simplices necessaries are selected manually, and require experimental tests to identify the appropriate values. For this data-set the specificity threshold is set to 1; this means that only classifier hubs with a 100% class-conditional probability can be selected.

Following this heuristic, only two classifier hubs are selected to represent the target concept; these are $\langle A0, A1 \rangle$ and $\langle B0, B1 \rangle$. These two classifiers

represent the correct target function, (A0 $\wedge$ A1) $\vee$ (B0 $\wedge$ B1), as any simplex will be considered to be of the target concept if it shares any of the two classifiers, $\langle A0, A1\rangle$ or $\langle B0, B1\rangle$. It is important to observe that these classifiers do not contain any of the irrelevant or correlated variables (I,K) initially present in the data-set. These have been filtered as "irrelevant" by the heuristic method. This idea of considering vertices that are not part of classifier hubs as irrelevant for the classification seems very powerful for identifying irrelevant variables in the data and is further studied in the following section.

3.6 Variable or feature selection by Q-analysis

As discussed in Section 3.3.2 the feature selection problem is that of identifying a sub-set of features or variables that provide discrimination information in relation to a classification task.

The previous section described how to use Q-analysis to find classifier hubs or relational concepts. As defined in Section 3.4.7, the vertices that are not shared by any of the classifier hubs describing the data, are considered irrelevant for the given classification. In the experiment above, variables I and K were not contained in any of the classifier hubs, and thus were irrelevant. To extend the results obtained in the previous experiment, a new experiment, based on the *iris* data-set collected by Anderson and published by Fisher [Fisher, 1936], is proposed in this section.

The iris data-set is composed of the measurement of four variables, namely, sepal width, sepal length, petal width and petal length of three different types of plant, 'setosa', 'versicolor' and 'virginica'. The complete data-set contains

50 instances of each plant having four measurements. The task consists of classifying each plant on the basis of their sepal and petal sizes. Figure 3-16 illustrates the iris data-set, where the vertical axis represents the sepal and petal sizes in centimeters, and the horizontal axis represents the plant instances ordered as follows. From 1 to 50 'setosa', from 51 to 100 'versicolor', and from 101 to 150 'virginica'.

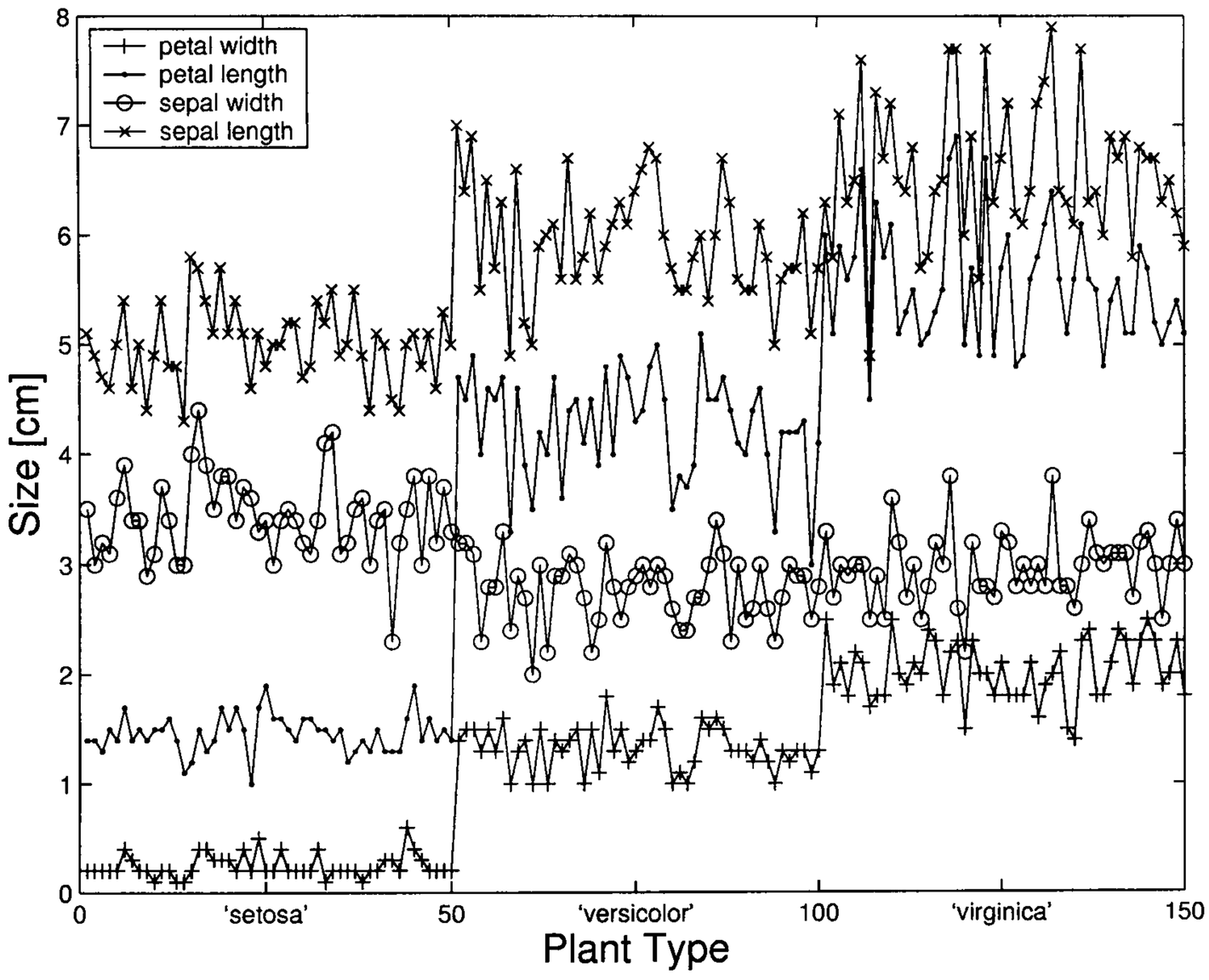


Figure 3-16: Iris data

Before classification, the complete iris data-set is divided into training and test data-sets. Each data-set contains half of the complete data, i.e., 75 instances of plants, 25 of each class. The experiment was conducted in three steps: (i) finding the classifier hubs for the training data-set, (ii) measuring the classifier's accuracy when classifying unseen plants from the test data-set, and (iii) studying the relevance of the classifier's vertices.

3.6.1 Simplex representation of the iris variables

In order to represent the continuous variables (see Figure 3-16) using binary values, each variable has been first normalised to a 0 to 1 range and then segmented into 10 equal intervals. This segmentation results in each continuous variable having 10 possible binary values. Let us refer to each of these binary values as: sw_i (sepal width), sl_i (sepal length), pw_i (petal width) and pl_i (petal length); each with $i = 1, 2, ...10$. Of the total of 40 binary values, only four at a time will be related to any given plant. In other words, each plant will be represented by a 3-simplex.

3.6.2 Star-hub analysis and classifier hubs

The star-hub analysis was used on the iris training data to discover its hubs. A total of 526 different hubs were found. The heuristic method presented in Section 3.5.2 was applied to the 526 hubs to find a set of classifier hubs that could explain the training data. In this case, the specificity threshold was also set to 1, thus only the hubs related to a unique class were considered as possible classifiers. Table 3.6 illustrates the resulting classifier hubs.

Table 3.6: Classifier hubs for the iris data

setosa			vesicolor			virginica		
classifier hub	spec	broad	classifier hub	spec	broad	classifier hub	spec	broad
$\langle pl_1 \rangle$	20/20	20/75	$\langle pl_5 \rangle$	8/8	8/75	$\langle pw_8 \rangle$	7/7	7/75
$\langle pw_1 \rangle$	19/19	19/75	$\langle pw_5 \rangle$	4/4	4/75	$\langle pw_9 \rangle$	6/6	6/75
$\langle pw_2 \rangle$	6/6	6/75	$\langle pl_4 \rangle$	4/4	4/75	$\langle pl_9 \rangle$	6/6	6/75
			$\langle sl_5, pl_6 \rangle$	6/6	6/75	$\langle pl_{10} \rangle$	6/6	6/75
						$\langle sl_3, pl_8 \rangle$	5/5	5/75

From the 526 hubs, the heuristic method selected only 12 classifier hubs. The relational concepts for the iris data-set were as follows:

$$\langle pl_1 \rangle \vee \langle pw_1 \rangle \vee \langle pw_2 \rangle \rightarrow setosa.$$

$$\langle pl_5 \rangle \vee \langle pw_5 \rangle \vee \langle pl_4 \rangle \vee \langle sl_5, pl_6 \rangle \rightarrow versicolor.$$

$$\langle pw_8 \rangle \vee \langle pw_9 \rangle \vee \langle pl_9 \rangle \vee \langle pl_{10} \rangle \vee \langle sl_3, pl_8 \rangle \rightarrow virginica.$$

The following section discusses how these relational concepts were used to classify the test data-set.

3.6.3 Classification using classifier hubs

The test data-set was used to measure the classification accuracy of the classifier hubs of Table 3.6. A plant is of a certain class if it shares a classifier hub related to that class.

Table 3.7: Classification results

	%correct	% unclassified	% misclassified
setosa	100	0	0
versicolor	80	20	0
virginica	80	20	0

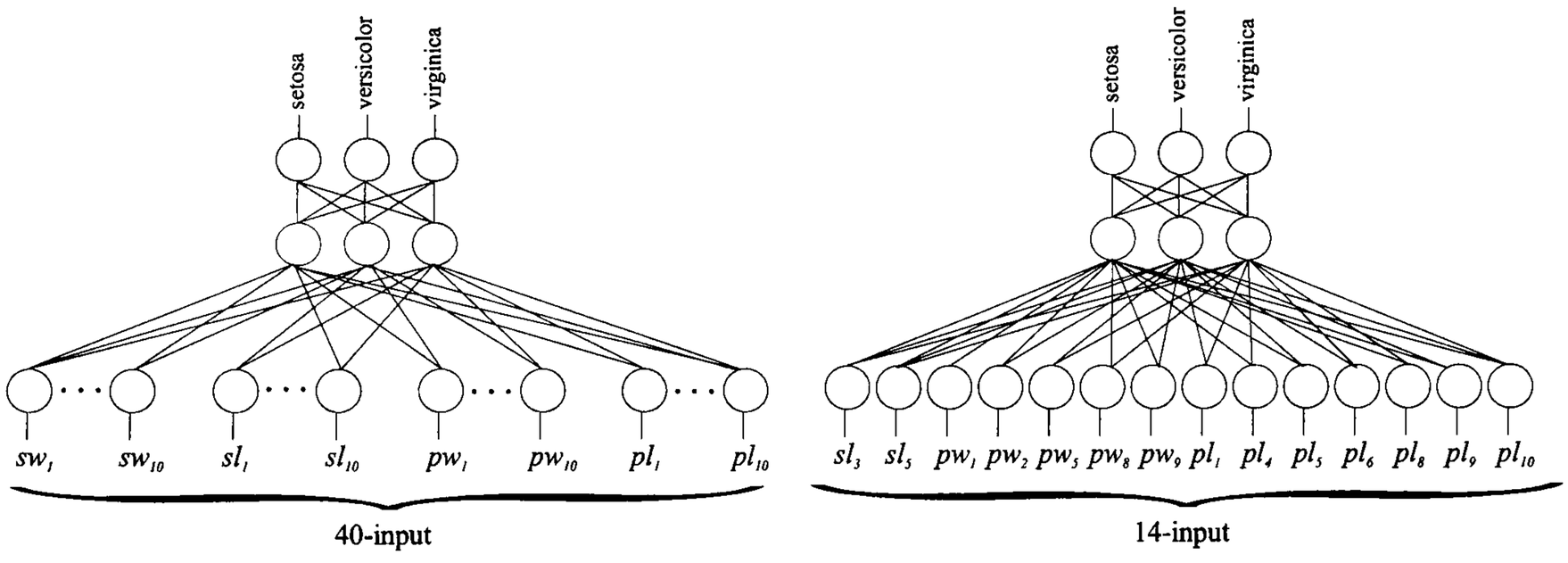
By applying the previous classifiers to the iris test data, we produced the results shown in Table 3.7, in which %correct, indicates the percentage of correctly classified primitives; %unclassified, indicates the percentage of unclassified primitives, i.e. primitives are considered unclassified if they do not share any of the classifier hubs; %misclassified, indicates the misclassified primitives, i.e. plants of a given class which share the hub of a different class.

The results in Table 3.7 indicate that a small number of classifier hubs, 12 in total, can be used to classify real-world data.

3.6.4 Study of variable relevance

In the experiment presented in Section 3.5 it was easy to validate whether the variables composing the classifier hubs were relevant or not, as the CorrAL data-set itself defines which variables are relevant, irrelevant or correlated. In the iris data-set there is no straightforward definition of which variables are relevant and which are not, thus in order to study their relevance the following experiment was conducted.

Two multilayer neural networks were used to classify the iris data, one using the complete set of variables (Figure 3.17(a)) and the other using only the variables that compose the classifier hubs of Table 3.6 (Figure 3.17(b)), both networks are illustrated in Figure 3-17.



(a) Network with 40 inputs

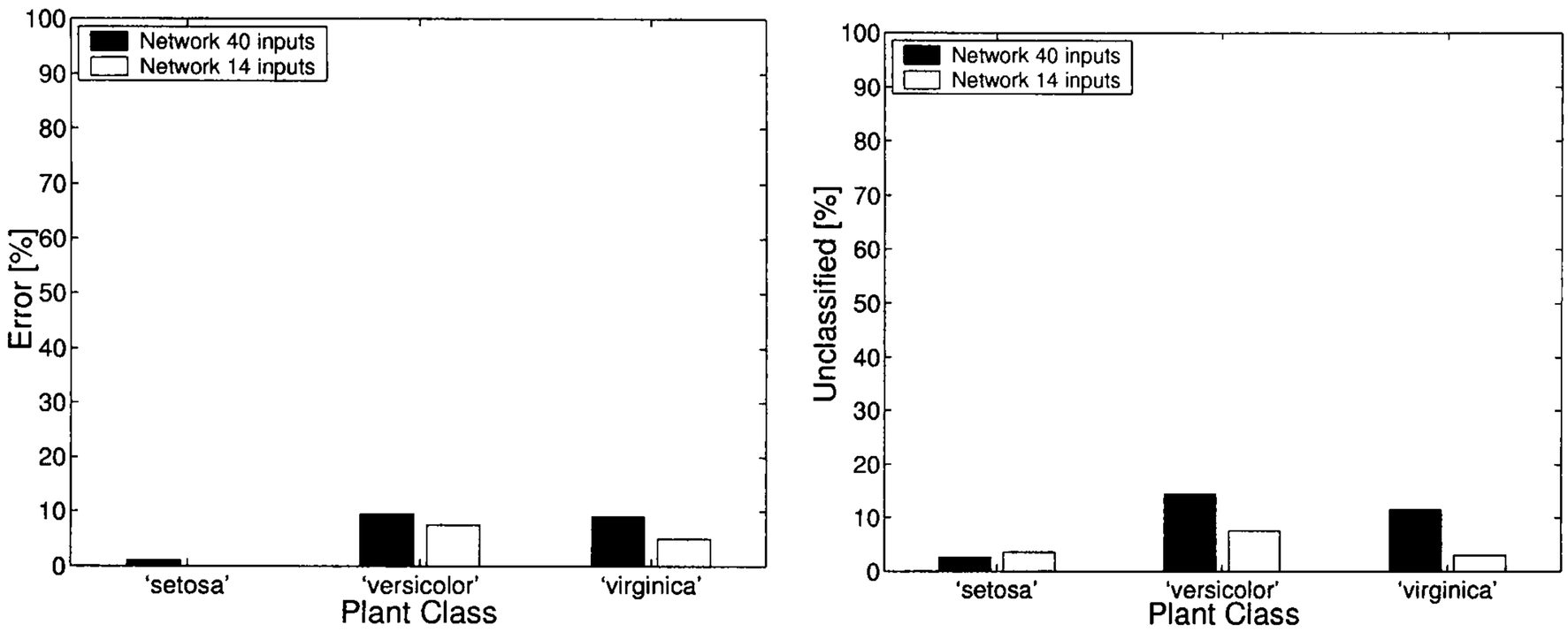
(b) Network with 14 inputs

Figure 3-17: Neural networks used for validation

The first network (Figure 3.17(a)) took as input the 40 variables described in Section 3.6.1. The second network (Figure 3.17(b))used only 14 variables

that appear in the classifiers hubs of Table 3.6, that is: (sl_3, sl_5 pw_1, pw_2. pw_5, pw_8, pw_9, pl_1, pl_4, pl_5, pl_6, pl_8, pl_9, pl_{10}). Both networks had 3 hidden neurons, 3 outputs, and were trained using backpropagation with 10 training examples of each class of plant.

The trained networks were tested five times against the remaining plants, and the average of their classification error was measured. The results of the classification are summarised in Figure 3-18.



(a) Misclassification error

(b) Unclassified error

Figure 3-18: Neural network classification error

In black is the result for the 40-input network; in white is the result of the 14-input network. Figure 3.18(a) illustrates the misclassification error and Figure 3.18(b) illustrates the unclassified error. These show that the 14-input network results in a slightly better classification accuracy. Although this accuracy is not significant to decide that the 14-input network outperforms the 40-input network, it can be said that both networks have similar accuracy. Thus, removing what the method considered as irrelevant variables does not decrease the network's prediction accuracy; this seems to indicates that those

variables were irrelevant in the first place.

It is also interesting to observe that most of the relevant variables in the classifier hubs are related to the *pl* and *pw* measurements. Only 2 out of 14 vertices correspond to the *sl* measurement and none correspond to the *sw* measurement. Thus, in some sense, the *pl* and *pw* measurements are more relevant than the *sl* and *sw* ones. If one looks at the plot of these measurements in Figure 3-16, the *pl* and *pw* measurements are more differentiated between plants than the *sl* and *sw* measurements. This indicates that the *pl* and *pw* are more relevant for differentiating between plants, which is consistent with the results of the experiment in this section.

3.7 Summary

This chapter developed the idea of a *concept* as the result of a classification process, in which specific primitives described in multidimensional spaces are mapped into general concepts or classes. The aim of building concepts in this thesis is one of creating a generalised and hierarchical representation of primitives.

Existent multidimensional data classification methods have been discussed, showing their strengths and limitations. It has been discussed that similarity metrics based on Euclidean distance can only be used if the mathematical properties of the multidimensional spaces are well-known. The feature selection problem has also been discussed, concluding that existing classification methods assume that it is the designer who identifies the relevant variables. Finally, the interpretability of hypotheses has also been described as a limitation of *black-box* classifiers, such as ANN, as it is difficult to analyse their

functionality.

In order to address these limitations, the theoretical methodology of Q-analysis has been presented, and a new technique for classification proposed, based on the idea of *classification hubs*. In order to demonstrate these technique two examples have been developed, one based on the synthetic CorrAL data-set, and the other on Fisher's Iris data-set. The experiments were based on supervised classification, that is, each primitive was labelled with its corresponding class. The aim in the experiments was to discover a set of classifier hubs (hypotheses) that appropriately classify the data-sets.

In the examples, it was shown first how the star-hub analysis could be used to find all the existing hubs in a set of data represented using simplices. Secondly, it was shown that a heuristic method could be used for selecting a subset of hubs and defining them as relational concepts. The heuristic method was based on selecting hubs that contained large number of simplices (hubs with high broadness) of a unique class (hubs with high specificity), until the subset of classifiers categorised the data with a sufficient accuracy. This approach proved applicable to the two data-sets, namely CorrAL and Iris.

Classifier hubs, or relational concepts in our approach, have been shown to be able to filter irrelevant variables used to describe primitives. That is, in the CorrAL data-set two variables, an irrelevant (I) and a correlated (K), were eliminated from the description of the target concept. In the Iris data-set, the vertices that appeared in the relational concepts were used as inputs to a multilayer neural network, which was compared with one using all the variables. The two neural networks had the same structure (except for the input units) and were trained with the same data. The results showed that

the network using only the filtered inputs, slightly outperformed in classifying unseen primitives with the network using all the variables. Thus, the method based on Q-analysis was again shown capable of filtering irrelevant variables in the data.

In summary, the novel classification method described here has the following characteristics:

- Similarity metric: In Q-analysis similarity is based on the structural relations of the data, i.e. in the q-nearness of simplices. As this similarity measure does not subsume the information in the data into a distance metric, the problem of comparing *incomparable* dimensions is alleviated.

- Feature selection: As it was shown, the classification method based on Q-analysis is very sensitive to the addition of variables or dimensions, this characteristic has been shown useful to detect irrelevant variables, and thus addressing the feature selection problem.

- Interpretability of hypothesis: As is was shown, and will be further seen in Chapter 6, because Q-analysis represents multidimensional data and their relations as binary value-attribute pairs, the resulting hypotheses are easy to interpret. This allows the designer to interact continually with the data and refine the methods and techniques being developed.

Chapter 4

A Multilevel Architecture based on Concept Generation

As seen in Chapter 2, the autonomy, robustness and flexibility of artificial agents can be improved by allowing them to adapt to their environment. Machine learning techniques can be used to provide flexibility and adaptive capabilities to autonomous systems.

A crucial aspect for the success of machine learning frameworks, such as reinforcement learning (RL), is that of the representation used to encode the target function. As seen in Section 2.4.2 applications with large state and action spaces require the representation to be general.

Chapter 3 introduced concepts as generalisations of primitives, showed some techniques for generating concepts, discussed their limitations and introduced the methodology of Q-analysis, and exemplified how a new classification technique based on Q-analysis addresses some of the limitations of previous classification techniques.

This chapter develops an architecture that allows the combination of ma-

chine learning frameworks, such as RL, with the idea of *concepts*. The underlying principle of this architecture is to generate concepts and use them as the representation upon which to base behaviour learning.

As seen in Section 2.2 architectures have two essential characteristics, namely their *structure* and *style*. This chapter presents the structural characteristics of the architecture, i.e. how the system is divided into sub-systems and their interaction. Some insights into the architecture's style are also given in this chapter, but Chapter 5 and Chapter 6, give more details about the computational processes underlying each sub-system.

4.1 Introduction

A major difficulty that arises in behaviour learning in multidimensional spaces is that of coping with high dimensionality of state and action spaces (see Section 2.4.1). This dimensionality makes the tabular representation of behaviour impractical in complex systems, such as robotic domains. In order to overcome these difficulties, 'generalised' representations of the target function or behaviour are necessary.

Concepts are, by definition, generalised classes composed of primitives (see Section 3.1.1); it is therefore conceivable, that in order to alleviate problems with dimensionality, one can divide behaviour learning into the following steps:

1. Generate concept representations.

2. Use concepts to learn behaviours or target functions.

The first step is one of learning a concept representation of states and actions, for subsequent learning of the target function based on the new representation. This thesis proposes to generate two types of concepts by: (i) taking the original state and action spaces, and generalising them by clustering primitives into concepts (in Chapter 5), and (ii) by generating relational concepts using Q-analysis (in Chapter 6).

The second step is a behaviour learning task, with the fundamental difference that behaviours are learned using concepts as representation instead of the original spaces.

These two steps are realised by two components of the architecture, namely, *concept generation* and *behaviour learning and control.* A structural description of the proposed architecture and its components is illustrated in Figure 4-1.

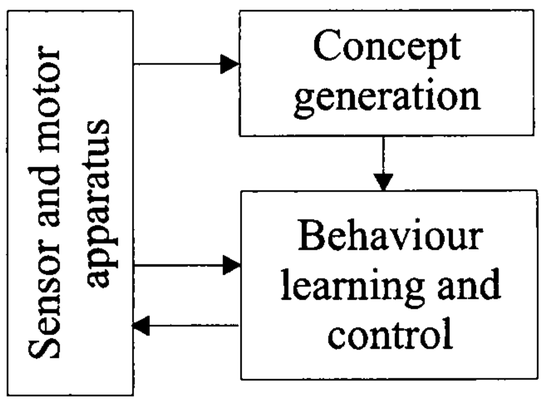


Figure 4-1: A structural description of the proposed architecture

The architecture has three main components:

- The *sensor and motor apparatus* component integrates all of the robot's sensors and motors, which directly interact with the environment. Sensors provide the state information from the environment, and motors provide the actions available to the robot.

- The *concept generation* component takes the state and action information from the *sensor and motor apparatus* component. This information is used to generate concepts relating to states and actions, namely, *state concepts* and *action concepts*. Different methods can be used to generate different types of concepts; Section 4.4 gives a detailed explanation of this component.

- The *behaviour learning and control* component takes the concepts formed by the *concept generation* and the information from the *sensor and motor apparatus*, and uses them to learn behaviours which are also used for controlling the robot. Section 4.5 gives a detailed explanation of this component.

The rest of this chapter develops one such learning architecture, where concepts are acquired and used as representations for behaviour learning. Learning can then exploit the generalisation and structuring that concepts impose on the original state-action spaces.

4.2 Arguments for a new architecture

The proposed architecture addresses the issue of developing a generalised representation which reduces the dimensionality problems of learning robotic behaviours. We first discuss why the existing methods to reduce dimensionality problems are not suitable for our purposes. We then discuss what are the novel characteristics of the proposed architecture.

4.2.1 Why not use other generalisation methods?

As reviewed in Section 2.4.2, some techniques based on generalisation exist to reduce the problems related to the dimensionality in behaviour learning. Generally, these methods are function approximators and multi-grid methods, and are aimed at representing a target function in a compact manner. For example, linear function approximators are used in combination with RL to represent the value function used for policy learning.

The reasons for not using any of those methods are the following:

- Function approximation methods do not generate a new and generalised representation of the robot's input space (state and action space), but generate a compact representation of the target function learned using those spaces. This has at least two implications:

  1. The approximations of functions using function approximators are 'flat' in the sense that, the approximator takes some inputs and approximates the output of the function without the usage of any explicit intermediate representations. This approach is successful for approximating simple functions, but would prove difficult at approximating complex ones. For example, having to learn the approximation of a value function corresponding to 'playing soccer' would be extremely difficult, as such value functions would depend on many sensory inputs or dimensions (player positions, player directions, player speeds, opponent strategy, team strategy, etc) and many actions (passing the ball to a particular player, dribbling to a particular position, getting in a particular strategic

configuration, shooting to goal in a particular direction, etc). In complex situations, it is helpful to generate multilevel representations from which complex representations can be achieved as the combinations of simpler ones.

2. The approximated functions are usually task dependent. For example, a common approach is to approximate the value function of an RL problem. As value functions are task dependent, the learned approximation can not be re-used in new tasks.

- Function approximation methods take some input variables or some features from these, and generate the approximation as combinations of the inputs and some internal parameters. For example, a robot could be approximating a value function as a linear combination of its sensory inputs, the actions selected in the environment, and some internal weights. The sensory information and the selected actions could also be input into a ANN where the combination of the input and the network's weights would approximate the function. This approach assumes that all of the robot's sensory inputs are relevant for the value function to approximate, thus the combinations of *all* the inputs is used to generate the output. In some situations, not all sensory inputs are relevant for the function to approximate. Thus, function approximators could be spending time and resources to approximate functions from irrelevant information. In extreme situations, many irrelevant variables could cause the approximator not to converge to the desired output.

- As function approximation methods are not capable of identifying and eliminating irrelevant variables or dimensions from the approximation,

they are prone to the curse of dimensionality. That is, the exponential growth of the resulting hyperspace of a function with many dimensions, even if some of these are irrelevant, will pose high computational requirements on the approximation method to learn the approximation.

- Multi-grid and variable resolution methods could, in principle, be used to develop explicit and general representations of the state or action spaces. These methods divide the state/action spaces or a value function into discrete hypercube regions. Each region represents a generalised region of the hyperspace. A drawback of multi-grid methods is that hypercubes are defined over *all* the dimensions (axes) of the hyperspace, even if these dimensions are not relevant.

4.2.2 Why this architecture?

The architecture developed in this thesis has the following characteristics:

- It can use different classification techniques to construct explicit and generalised representations, known as *concepts*, of the state and action spaces. These concepts are then used as generalisations for behaviour learning. Thus, the architecture is capable of learning in large state or action spaces, representing the target function in a generalised manner.

- Concepts are defined within a multilevel hierarchical representation, that is, concepts at lower-level descriptions can be treated as primitives and used to define concepts at higher-level description. Thus, it is possible to learn concepts from low-level sensor and motor data and scale up to complex representations.

- Concepts are defined as hypercubes in multidimensional hyperspaces. Although this is similar to the representations used by multi-grid methods, it is possible to define hypercubes which ignore irrelevant dimensions or axes of the hyperspace.

- A classification method based on Q-analysis, is presented within the architecture which is capable of identifying irrelevant dimensions in relation to a concept, thus eliminating unnecessary dimensions and reducing the problems related to the curse of dimensionality.

- Concepts are grounded in sensory and motor data. Thus, symbol grounding problems are alleviated.

This section motivated the need for a new architecture for behaviour learning by presenting the main shortcomings of using function approximation methods as generalisations. It also presented the main characteristics of the proposed architecture. The next section gives an overview of this architecture.

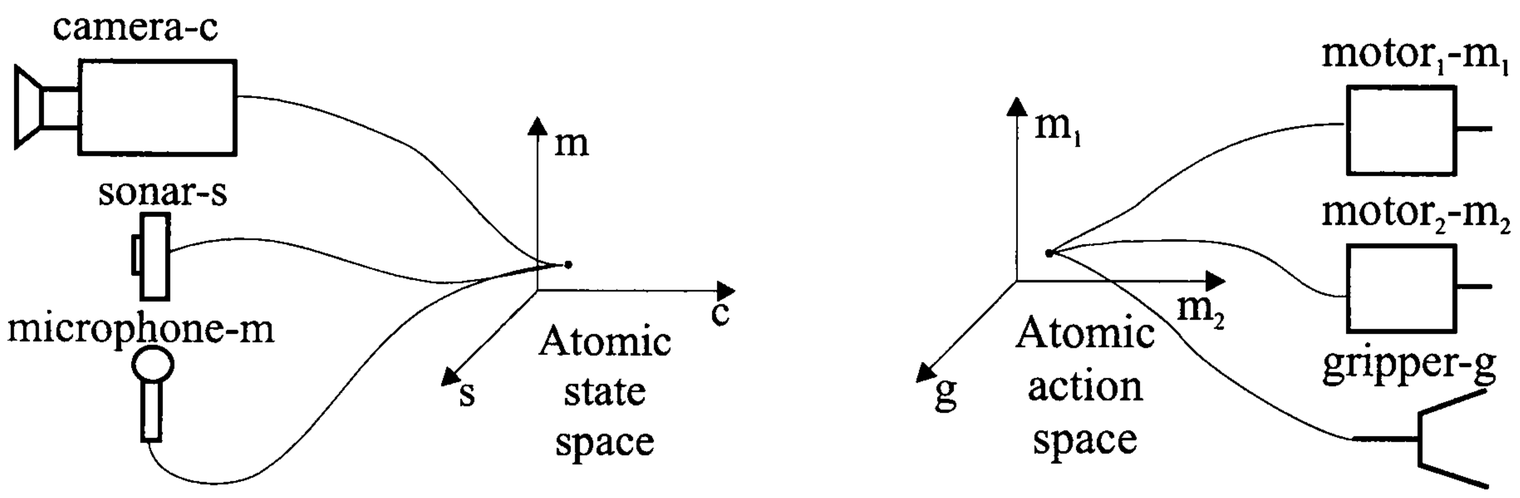
4.3 Architecture overview

Similar to the RL framework (Section 2.3.2), the architecture presented here is based on the definition of a set of observable environmental states S, and a set of possible actions A. Time is assumed to be discrete and to increment in constant intervals.

The state of the environment is perceived through the robot's sensors. Thus, the environmental state space S is equivalent to the robot's sensor space. An action is defined as the combination of all the possible commands

that can be sent to the robot's motors. Thus, the action space A is equivalent to the robot's motor space.

As the architecture presented here develops a multilevel representation of S and A, let us refer to the states and actions defined at the lowest possible description-level as *atomic*. That is, atomic states and actions are those formed directly from the robot's sensor and motor values, respectively. Figure 4.2(a) illustrates a robot's sensory apparatus, and how the atomic state space is formed by it. Figure 4.2(b) illustrates a robot's motor apparatus, and the atomic action space related to it. This characteristic of defining atomic states and actions directly from the robot's sensor and motor data, will allow the multilevel representation to *ground* concepts on sensor and motor data.



(a) Atomic states (b) Atomic actions

Figure 4-2: Atomic states and actions

To describe the architecture let us use the robot in Figure 4-3 as an example. The robot has two motors, m_1 and m_2, connected to the right and left wheels, and two sensors x_1 and x_2, which measure the distance d and angle α with respect to an object. Let us assume that the response of sensor x_1 is $r(x_1) \in 0 \ldots 100\ cm$ and that $r(x_2) \in 0 \ldots 360^o$. Furthermore, let us also assume that the activation of motor m_1 is $a(m_1) \in 0 \ldots 100\%$ and that $a(m_2) \in 0 \ldots 100\%$.

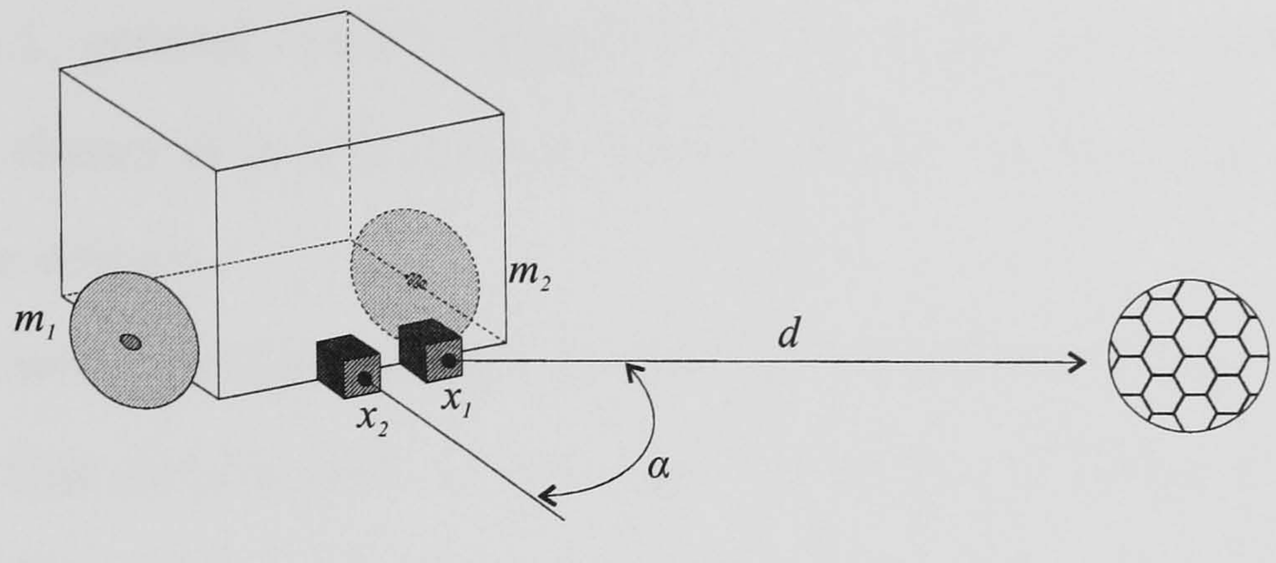
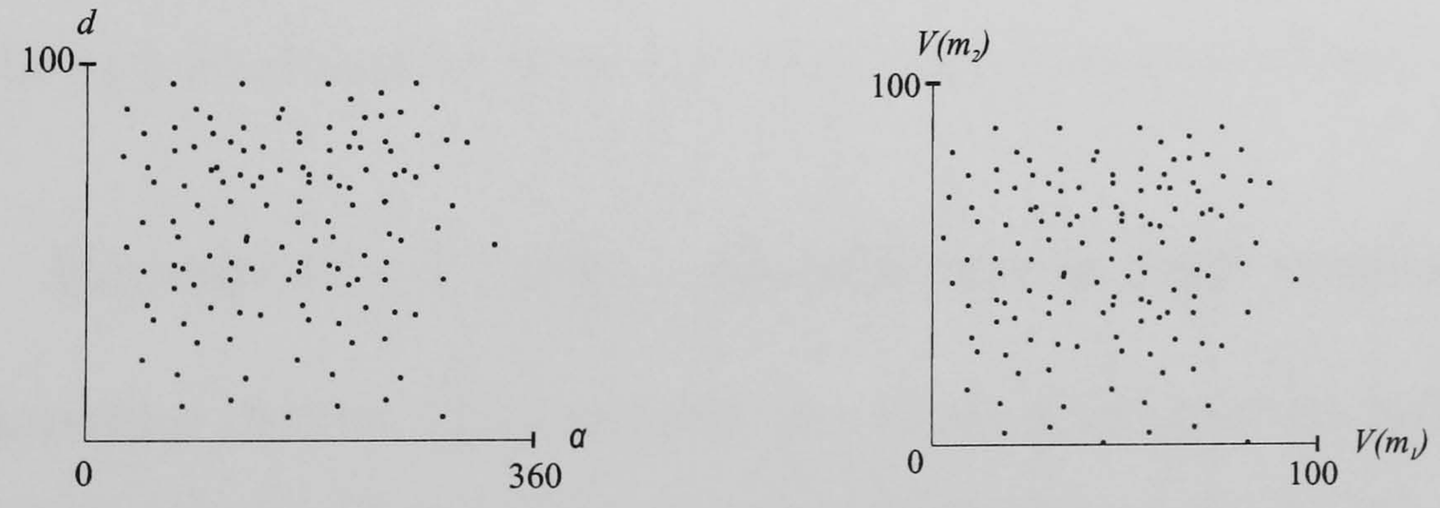


Figure 4-3: A generic mobile robot

The combination of the responses of the two sensors constitutes the atomic state space for the robot. Figure 4.4(a) illustrates this state space (sensor space) as a two-dimensional coordinate space, where the axes represent the response of the sensors. Atomic states are represented by dots. In a similar manner, the combination of motor activations constitutes the robot's atomic actions space. Figure 4.4(b) illustrates this action space (motor space). The axes represent the activation value for each motor. Atomic actions are represented by dots.



(a) State space

(b) Action space

Figure 4-4: State and action spaces for the generic robot

Using this common definition of state and action spaces, the architecture defines a hierarchy of state and action concepts. Concepts are, as defined in

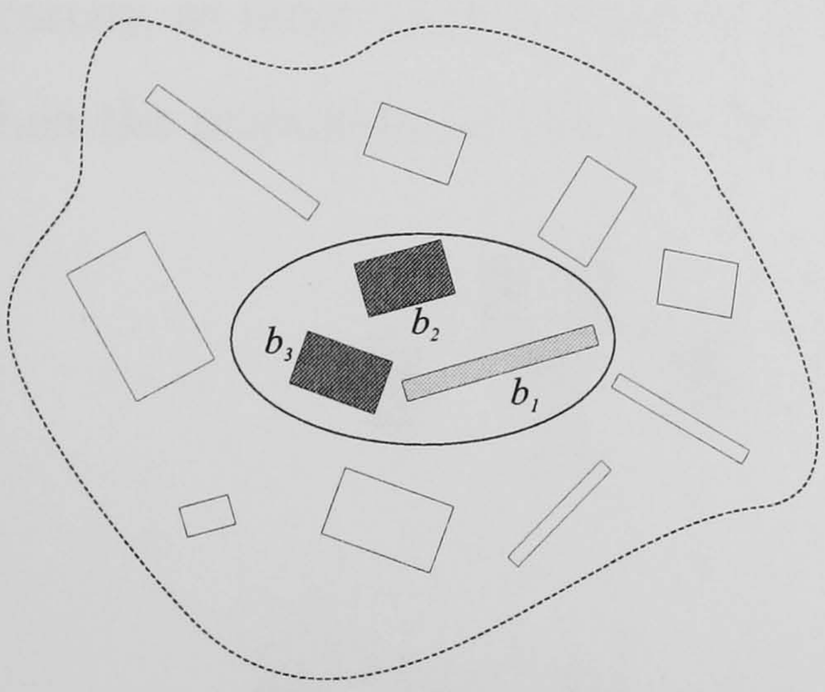
Section 3.1.1, general classes composed by primitives, thus, state concepts are general classes of primitive states, and action concepts are general classes of primitive actions.

The following section describes the idea of *hierarchical classification* which is used in this architecture to represent *concepts* at different levels of description. The motivation for introducing hierarchical classification is the following. Learning a complex robotic behaviour, such as playing soccer, represented by using atomic states and actions, would be a difficult task given the many possible state and action combinations. As a matter of comparison, it would be like teaching a young child to play soccer in terms of the length, direction, speed of its steps and the distances and angles to objects such as the ball, opponents and goal. In this case, decomposing the complex behaviour into a hierarchy of simpler sub-behaviours can facilitate learning [Stone, 1998, Dietterich, 1998, Barto and Mahadevan, 2003]. For example, the playing soccer behaviour can be decomposed into sub-behaviours, such as dribbling, ball passing, pass selection, etc. Then learning the playing soccer behaviour as a combination of the sub-behaviours is simpler [Stone, 1998].

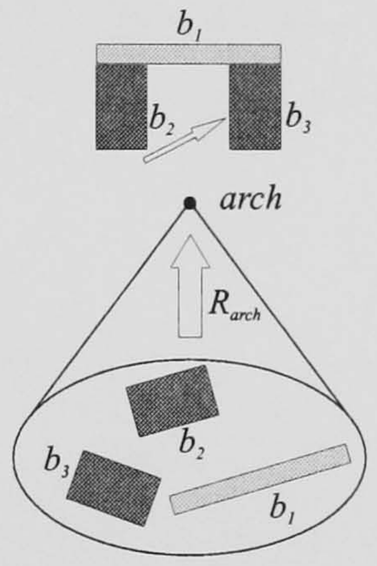
4.3.1 Hierarchical lattice classification and concepts

Let us introduce the idea of hierarchical classification using lattice hierarchies by following a simple example. Figure 4.5(a) illustrates a space of *blocks*, from which the *Euler ellipse* selects the set of blocks: $\{b_1, b_2, b_3\}$. The parts of this set can be arranged in a spatial configuration, such as the one illustrated in Figure 4.5(b). As previously used in classification, let us refer to the parts forming configurations as primitives and the resulting configurations as

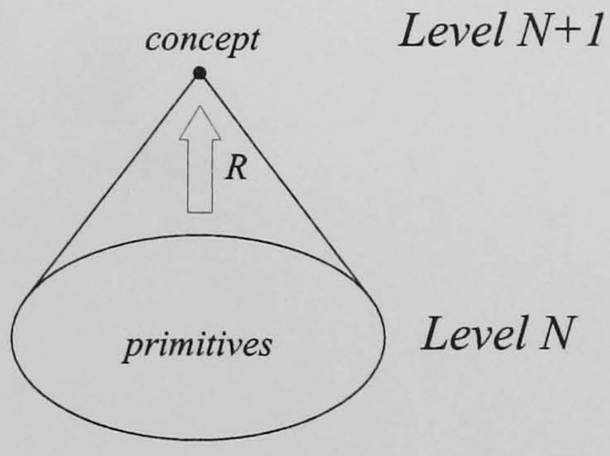
concepts, this leads to a hierarchical notation where primitives are considered to be at description level *Level N* and concepts at level *Level N + 1*. This notation is illustrated in Figure 4.5(c). The resulting concept in Figure 4.5(b) follows a special relation of its primitives, namely, R_{arch}. That is, blocks b_2 and b_3 are horizontally placed at a certain distance from each other, and b_1 is vertically placed on top of b_2 and b_3.



(a) Space of possible blocks



(b) Blocks classified into an arch



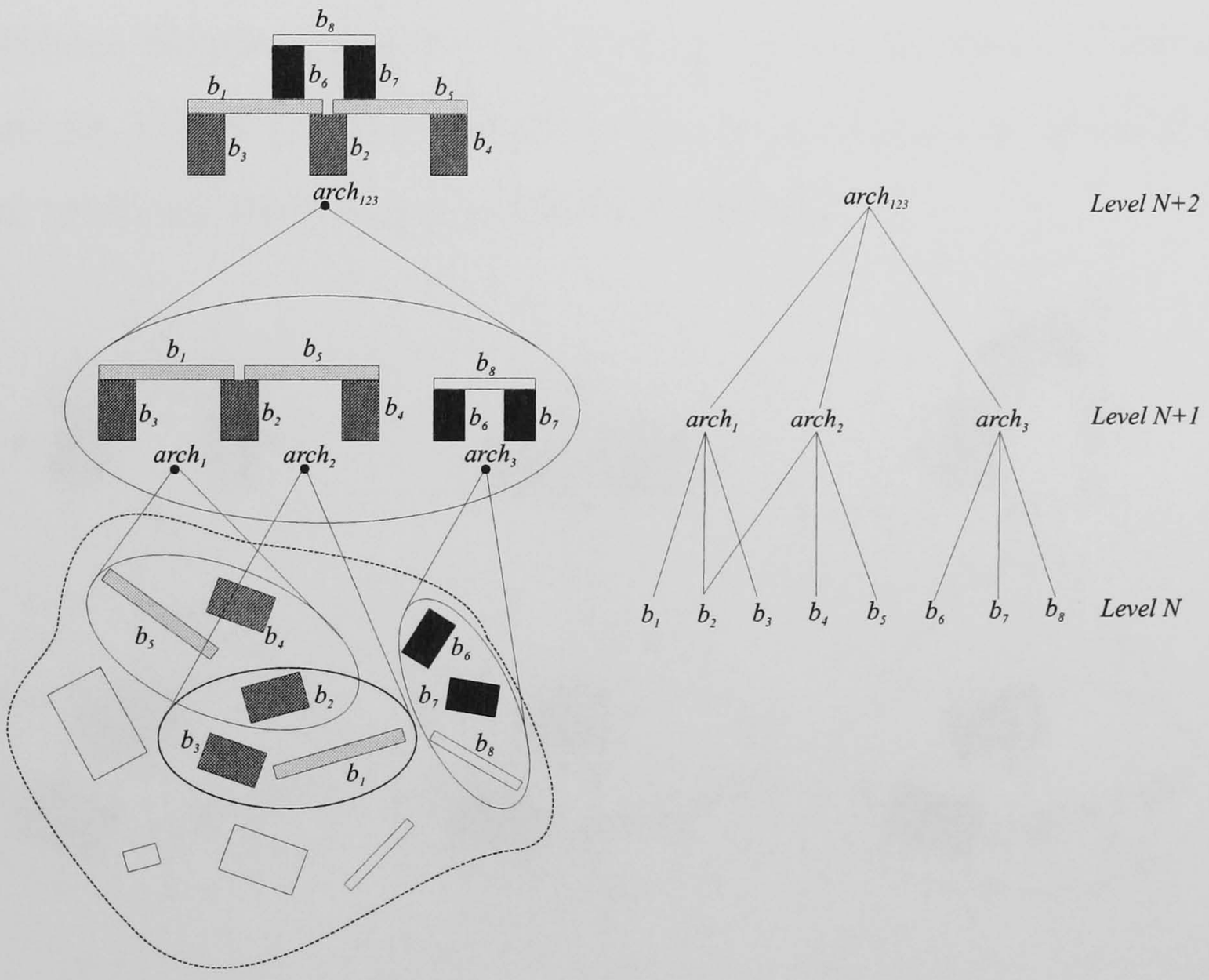
(c) Hierarchical notation

Figure 4-5: Classification of a set of primitives into a concept

An important aspect of classifying parts into wholes is that the resulting concepts may have emergent properties. For example, the *arch* in Figure 4.5(b) has the emergent property of allowing other objects to pass through

it, which is not a property of any of its individual parts. If the resulting concept has interesting properties, it can be given a name, such as *arch*. This process of giving a name is one of making the concept explicit, and does not occur in function approximation methods.

A multilevel representation considers that, if primitives are at description level *Level N*, then the concept is at description level *Level N* + 1 in the hierarchy, as illustrated in Figure 4.5(c). If N is the lowest-level of description, then the primitives at this level are atomic.



(a) Blocks classified into arches

(b) Lattice hierarchy

Figure 4-6: Example of a multilevel lattice hierarchy

A characteristic of multilevel hierarchies is that concepts can be recombined, as if they were primitives, resulting in higher-level concepts. Figure

4-6 illustrates a multilevel hierarchy, where blocks at level *Level N* are combined to produce arches at level *Level N+1*, and arches at level *Level N+1* are combined to produce further arches at level *Level N+2*. When a primitive is part of more than one concept, for example b_2, which is used by $arch_1$ and $arch_2$, then the hierarchy is known to have a *lattice* structure or to be a *lattice hierarchy.*

4.3.2 Relations between primitives in classification

The relations between primitives have effects on the definition of concepts. For example, Figure 4-7 illustrates how a set of blocks can be classified with different relations which result in different concepts.

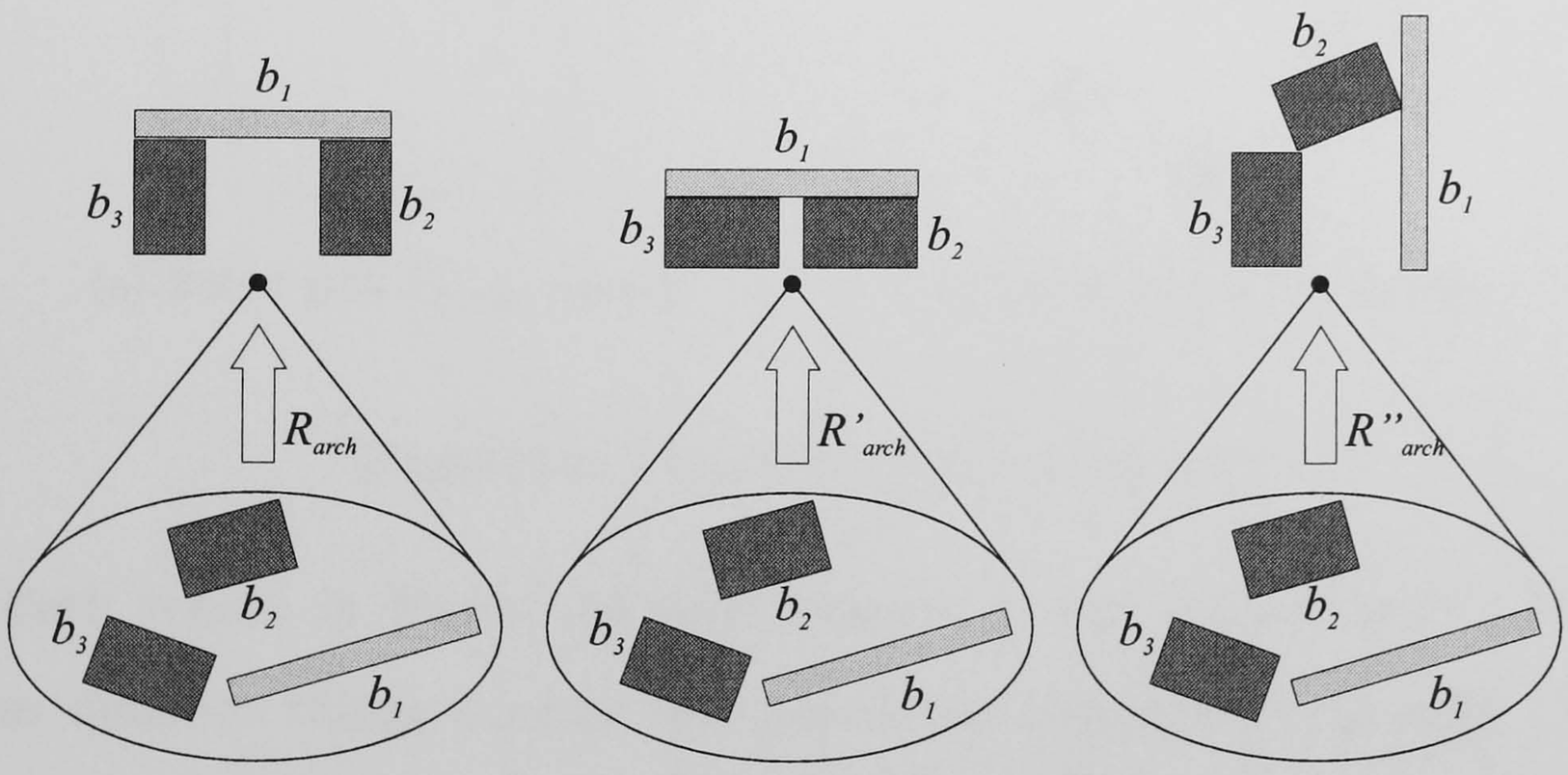
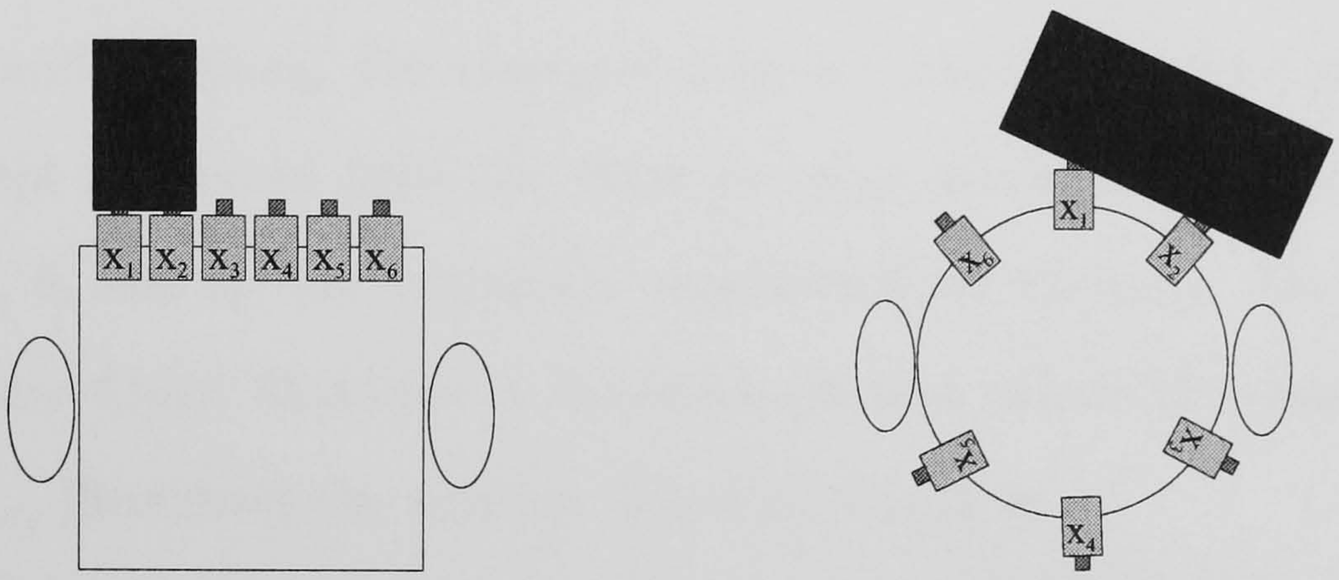


Figure 4-7: A set classified with different relations

This means that it is not just the set of primitives that gives rise to a concept, but also the particular relationship. As illustrated in the figure, the relation R_{arch} has supporting blocks vertical, while R'_{arch} has them horizontal. When it is desirable to distinguish the relation that defines a simplex, we can

make it explicit with the notation $\langle b_1, b_2, b_3; R_{arch} \rangle$. It can then be discriminated from $\langle b_1, b_2, b_3; R'_{arch} \rangle$, even though the underlying set of primitives is the same for both.

Relations between primitives can be of various types. For example, in robotics the relation could be given by the physical construction of the robot. Let R_{array} be the relation between the sensors of the robot illustrated in Figure 4.8(a) and let R_{ring} be the relation between the sensors of the robot illustrated in Figure 4.8(b).



(a) Robot with R_{array} sensors (b) Robot with R_{ring} sensors

Figure 4-8: Physically different robots

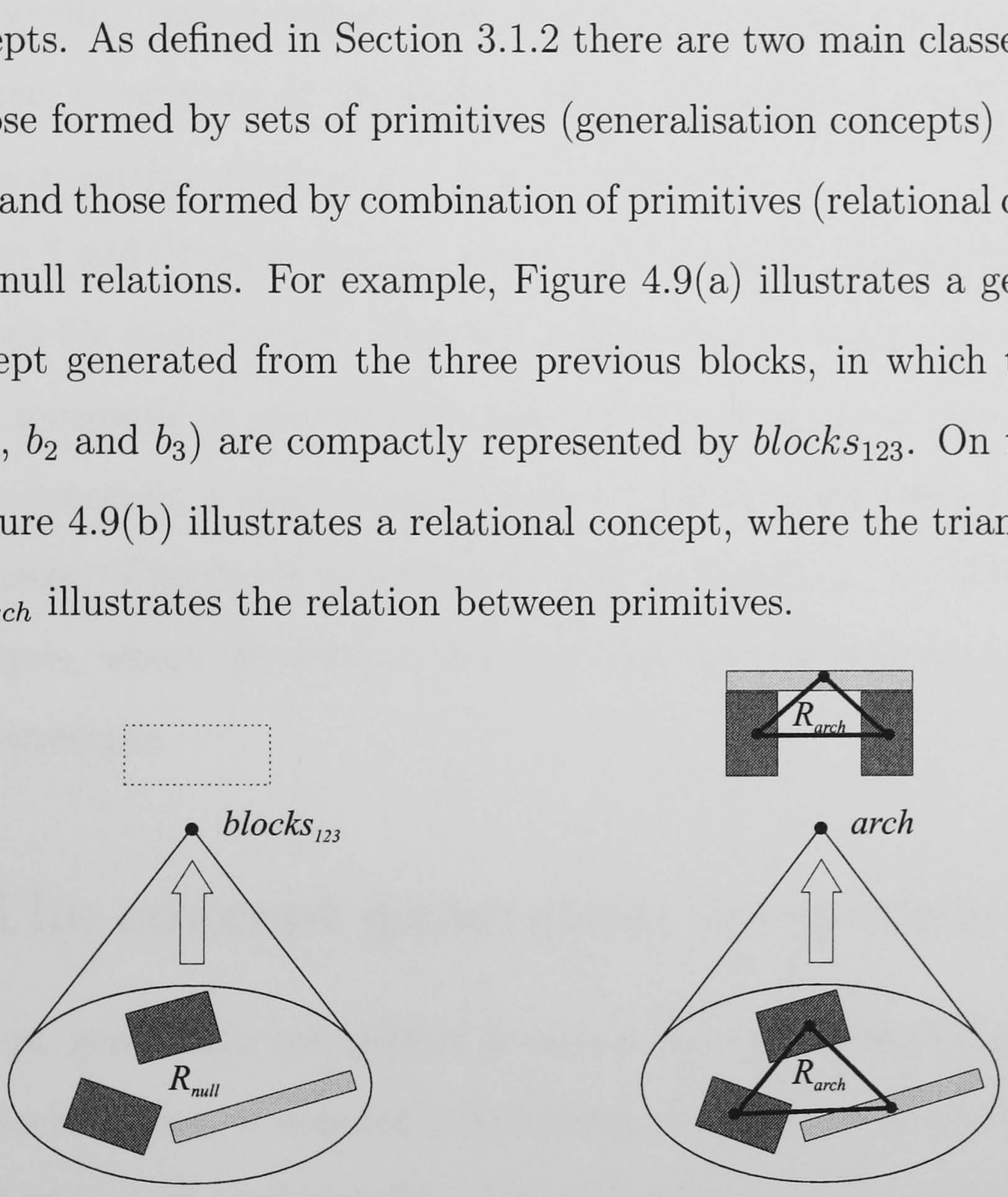
Both robots in Figure 4-8 have sensors x_1 and x_2 activated. These mean different things because the robots are assembled differently. The previous notation allows representation of the following: $\langle x_1, x_2, r_{array} \rangle \neq \langle x_1, x_2, r_{ring} \rangle$.

Other relations include *temporal relations* which indicate that sensor states happening in different order have different meaning. For example, for the robot in Figure 4.8(a) the activation sequence ($x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5 \rightarrow x_6$) indicates that an object is moving from left to right, whereas the sequence ($x_6 \rightarrow x_5 \rightarrow x_4 \rightarrow x_3 \rightarrow x_2 \rightarrow x_1$) indicates that the object

is going from right to left. Thus, the same primitives $x_1, x_2, ...x_6$ form two different concepts if the temporal relation is taken into account.

4.3.3 A unified view of concepts

The relations between primitives have an important role in their classification into concepts. As defined in Section 3.1.2 there are two main classes of concepts: those formed by sets of primitives (generalisation concepts) with null relations, and those formed by combination of primitives (relational concepts) with non-null relations. For example, Figure 4.9(a) illustrates a generalisation concept generated from the three previous blocks, in which the three blocks (b_1, b_2 and b_3) are compactly represented by $blocks_{123}$. On the other hand, Figure 4.9(b) illustrates a relational concept, where the triangular relation R_{arch} illustrates the relation between primitives.



(a) Example of a generalisation concept (b) Example of a relational concept

Figure 4-9: Example of a generalisation versus a relational concept

In the architecture presented in this thesis, both concepts are used. As will be shown in the following chapters, generalisation concepts provide the

means to generate compact representations of primitives, thus reducing the size of the space formed by primitives. Relational concepts can have various applications depending on the relation between primitives. For example, a relational concept formed by primitives and their temporal relation of occurrence, encodes the *history* of the transition of primitives. A relational concept encoding such a history may be useful for representing behaviours. For instance, a sequence of decreasing distances towards a target could be encoded as a 'getting closer to the target' behaviour.

Chapter 5 and Chapter 6 of this thesis experimentally validate two different methods for classification. Chapter 5 experiments with a distance based clustering technique to generate concepts. These type of concepts are generalisation concepts, a unique primitive being sufficient for being associated with a cluster. Chapter 6 experiments with a classification method based on Q-analysis, where the relation between primitives is necessary to define relational concepts.

4.4 The concept generation component

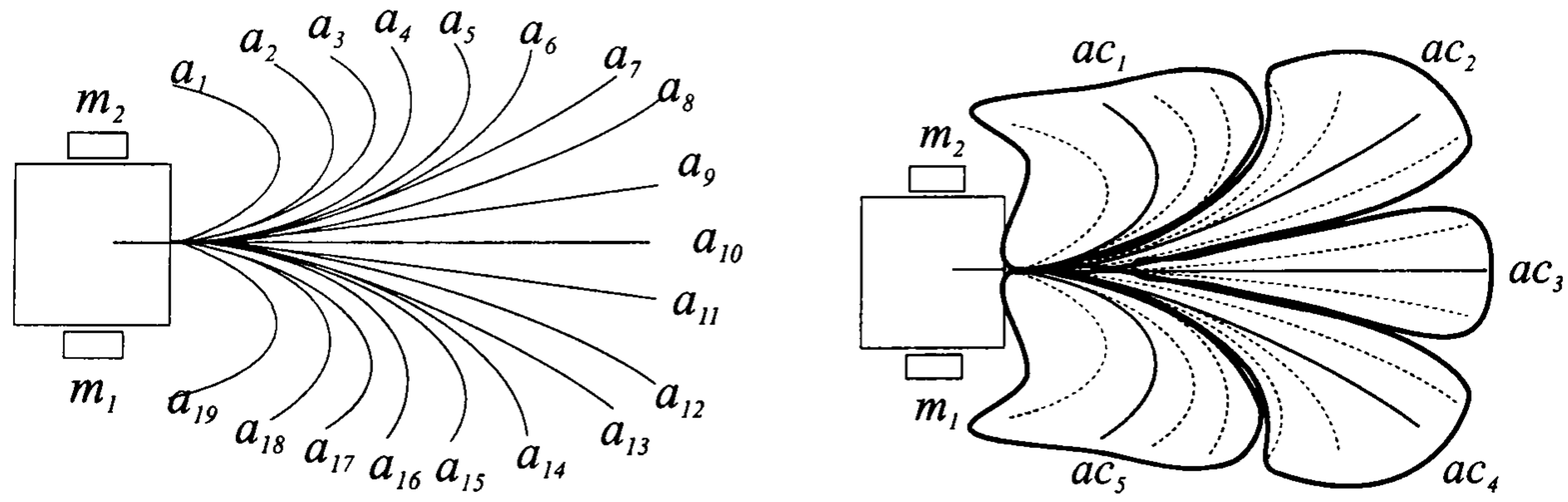
The concept generation component is one of the main components of the proposed architecture. It uses the information coming from the robot's sensor and motor apparatus, and classifies this information into hierarchies of state and action concepts. The next sections explain this process.

4.4.1 Hierarchical action classification

Action concepts are classes defined over the atomic actions available to the robot (motor space). Let us describe action concepts by means of an example. The robot in Figure 4-3 had the action space illustrated in Figure 4.4(b). This action space has $|A| = 100 \times 100 = 10^4$ atomic actions. The next section illustrates how to generate action concepts given atomic actions. As discussed in Section 3.1.2, there exist two main types of concepts, generalisation and relational. The next sections explain how generalisation concepts and relational concepts can be acquired from a robot's motor data.

Generalisation action concepts

If the robot selects a particular atomic action for a certain period of time, the interaction between the robot and the environment results in the robot moving, thus describing a *trajectory*. Figure 4.10(a) illustrates some possible trajectories, where each corresponds to an atomic action, a_i, being selected for a period of time. As the number of atomic actions is large (10^4), the number of resulting trajectories will also be so.



(a) Some possible trajectories (b) 'Similar' trajectories

Figure 4-10: Robot trajectories and groups of 'similar' trajectories

With real robots, the same atomic action will hardly ever result in exactly the same trajectory; this is because of the chaotic behaviour of robotic systems. Thus, in this context, having precise definitions of trajectories is not necessarily useful, as in many cases these will not be attainable. Therefore it seems logical to define trajectories in a more relaxed manner. For example, atomic actions can be grouped according to the similarity of the trajectories they produce. Figure 4.10(b) illustrates this process, where similar trajectories are grouped, and the actions that lead to these groupings are defined as *action concepts*, ac_i.

Effectively, what is happening in Figure 4-10 is: $ac_1 = \langle a_1, a_2, a_3, a_4, a_5, R_t \rangle$, where R is null. In other words any of the primitives is sufficient to define the concept. In Chapter 5, it will be shown how distance based clustering techniques can be used to generate generalised action concepts related to motor commands. That is, sets of similar motor commands are clustered into generalisation concepts of atomic actions.

Relational action concepts

Following the multilevel methodology, the actions concepts defined at level *Level* $N+1$ can be further recombined into action concepts at level *Level* $N+2$, and so on. This idea is illustrated in Figure 4-11, where atomic actions are at level *Level* N, and action concepts from level *Level* $N+1$ upwards. The arrows in the figure represent the trajectories resulting from each action or action concept, and as can be observed, higher levels in the hierarchy correspond to more complex trajectories. From level *Level* $N+1$ upwards, action concepts are defined with temporal relations. That is, actions are ordered

by an execution order. Changing this relation (order of execution) results in different trajectories at higher-levels. In our terms, these are relational concepts.

Action concepts generated by taking into account temporal relations are similar to *plans*. That is, a sequence of n ordered atomic actions ordered by their time of execution, $\langle a_1, a_2, \ldots, a_n; R\rangle$ is a plan of n steps. For example, the action concepts at level *Level* $N+3$, in Figure 4-11, can be seen as four step plans as they are composed of four atomic actions.

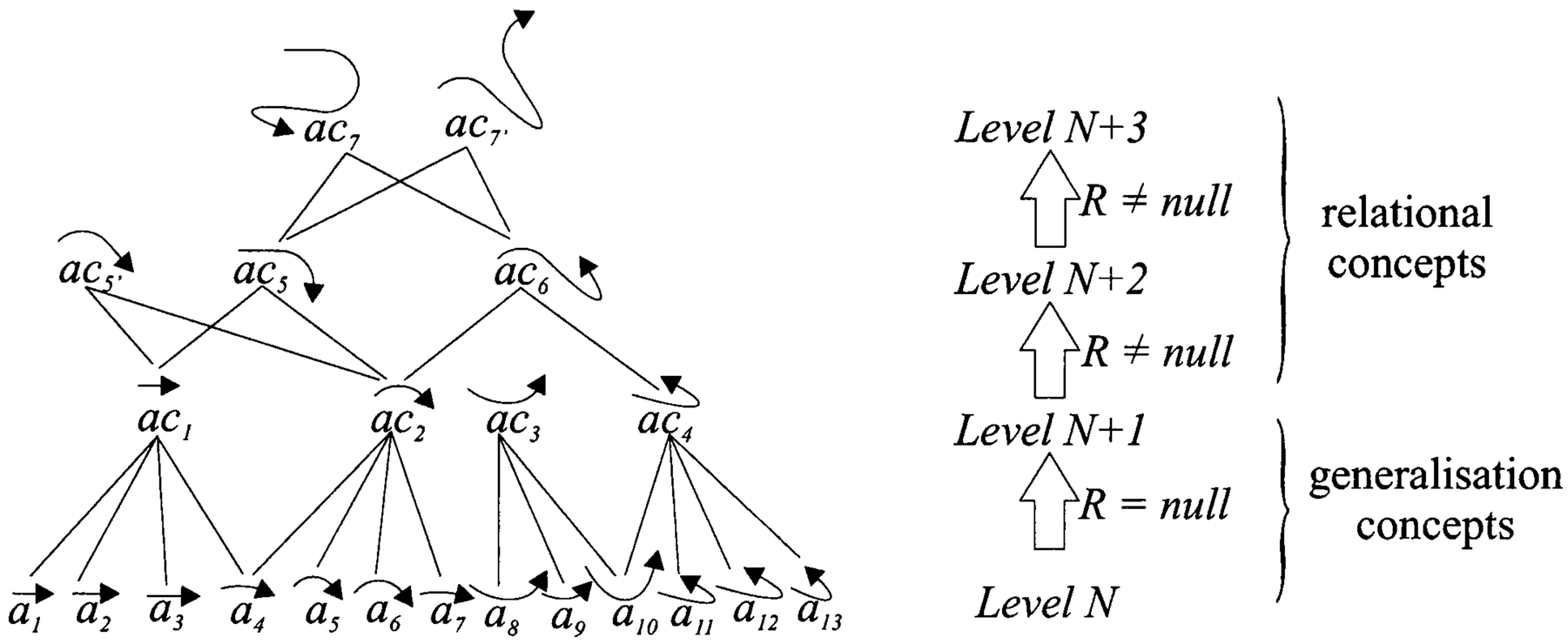


Figure 4-11: Hierarchical classification of action concepts

4.4.2 Hierarchical state classification

State concepts are concepts defined over the atomic states that a robot can perceive (sensor space), such as the state space of the robot in Figure 4-3 and illustrated in Figure 4.4(a). The next section describes how generalisation concepts and relational concepts can be generated from the robot's sensor data.

Generalisation state concepts

Following the previous example, at any given time the robot perceives an atomic state s which is determined by the value of the distance sensor x_1, and the angle sensor x_2. In this case, $s = \{d, \alpha\}$. These values are illustrated in Figure 4-12.

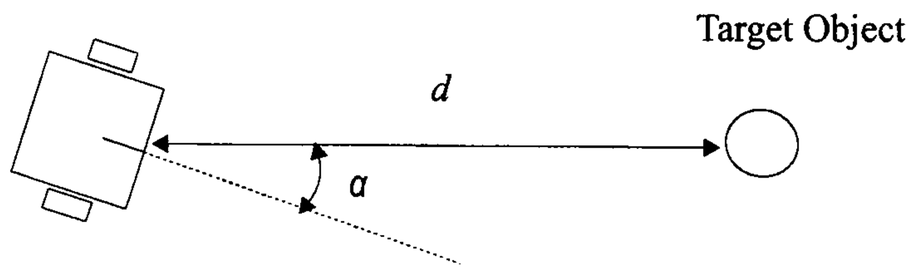


Figure 4-12: Atomic state

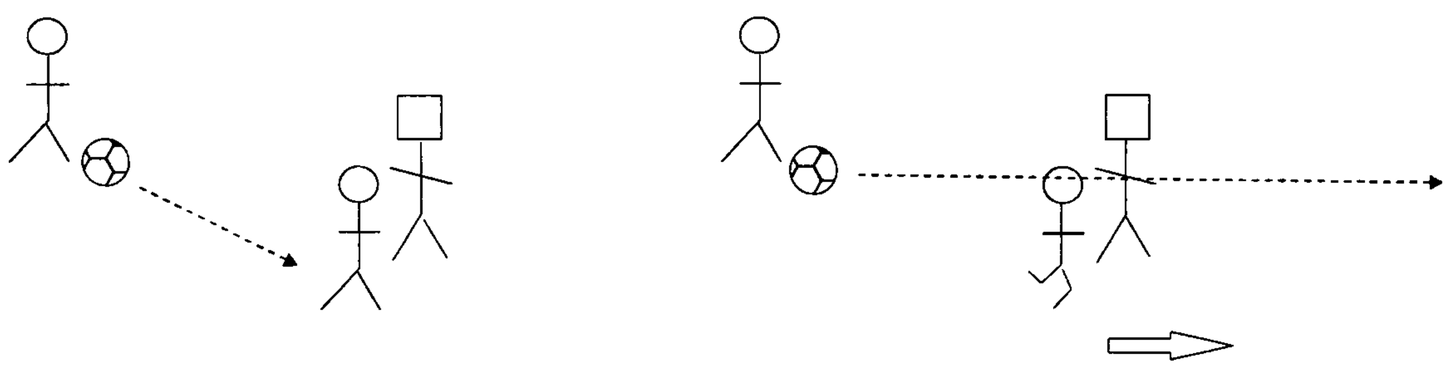
Atomic states can be classified into *state concepts* at higher description levels. If a state concept at level *Level* $N + 1$ is the result of a state classification with a null relation in its primitives, then these state concepts are generalisations of their primitives. As stated previously, atomic states can be clustered into generalisation concepts. Chapter 5 demonstrates how this can be done.

Relational state concepts

State concepts can also be classified with non-null relations. For example these relations could be based on the simultaneous occurrence of some sensor values, $x_1 = a$ and $x_2 = b$ and $x_3 = c$. The relation could be different if one of them is missing.

For example, the players in Figure 4.13(a) and Figure 4.13(b) are in the same positions with respect to each other, but the two situations are very different. In Figure 4.13(b) the team-mate player is running towards

an empty area while in Figure 4.13(a) it is static. Thus, the notion of the player's positions in combination with its speed of movement is different from when only their positions are measured. Thus it can be said that: $\langle positions, speeds \rangle \neq \langle positions \rangle$.



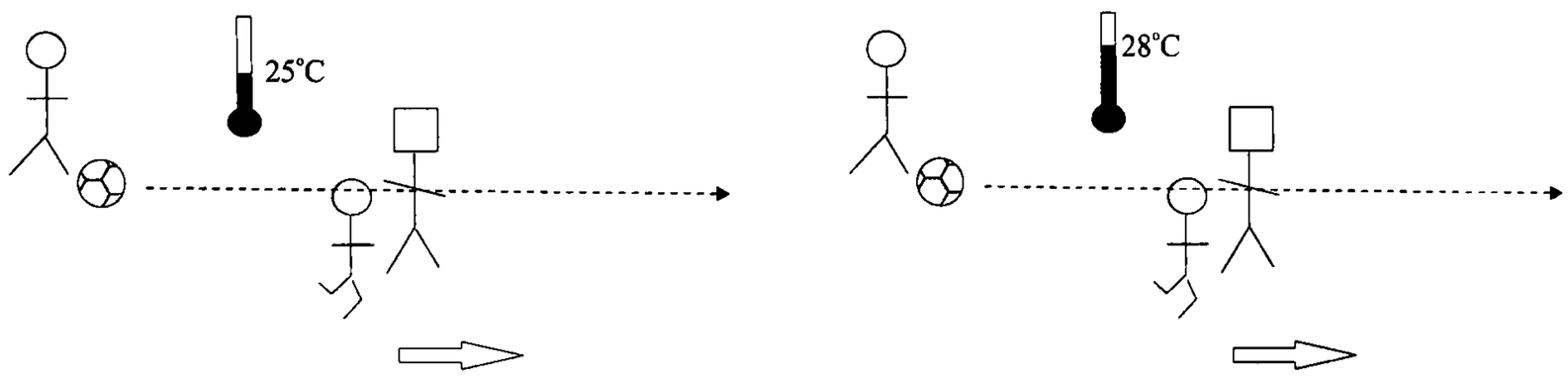
(a) Static players (b) Team-mate running to an open area

Figure 4-13: Soccer players in the same positions but in different situations

An obvious way of taking all the sensor relations into account is to consider logical *AND* operations between them, that is, the combination of $x_1 \wedge x_2 \wedge x_3 \ldots \wedge x_n$, can be used to define the relational concepts generated by n sensors. This is the approach usually taken when the state of an RL robot is described as the *AND* combination of *all* its state variables. The problem with this approach is that in many cases not all the sensors provide relevant information. These irrelevant sensors only increase the number of possible combinations.

For example, Figures 4.14(a) and 4.14(b) illustrate a similar relationship between soccer players, that is, if the variable temperature is not taken into account then both situations would be identical. On the other hand, if the temperature variable is taken into account, the two situations become different. The issue is then to asses whether the temperature variable affects the situation enough to be considered as part of the concept describing the situation. That is should one use, $\langle positions, speeds, temperature \rangle$ or

$\langle positions, speeds \rangle$?



(a) Situation low temperature (b) Situation higher temperature

Figure 4-14: Soccer players in similar situations discriminated by a temperature variable

Chapter 6 experimentally demonstrates, on robotic data, the classification method described in Chapter 3 which defines relational state concepts and gives a possible answer to how assessing the relevance of variables.

4.5 The behaviour learning and control component

The *behaviour learning and control* component is in charge of learning behaviours represented by concepts, and using these behaviours to control a robot.

4.5.1 Behaviour learning

In this architecture, as in RL, behaviour learning is based on adapting a mapping between state concepts and action concepts, namely the *behaviour function* (or policy in RL), B. In principle, it would be possible to define a reward value, and modify some existing RL algorithm to learn behaviour

functions that maximise the reward based on the hierarchical description of states and actions. For example, if action concepts are defined as an ordered set of atomic actions, then it would be necessary to treat the RL problem as a *semi-Markov decision process*, as action would be *time extended*. Moreover, the termination condition of an action concept should be based on the satisfaction of state concepts. We believe that an RL framework such as the *options framework* [Sutton *et al.*, 1999] would be a good candidate to use within the proposed architecture.

Although the previous approach seems possible, this thesis exploits a method for learning behaviours based on learning by example (supervised learning). The reasons for choosing a supervised approach are: (i) simplicity of implementation in comparison to RL methods, and (ii) the main focus of this thesis is to demonstrate that an architecture based on multilevel representation of concepts can be used for learning behaviours and controlling robots, and thus the method used to learn these behaviour is not of central relevance.

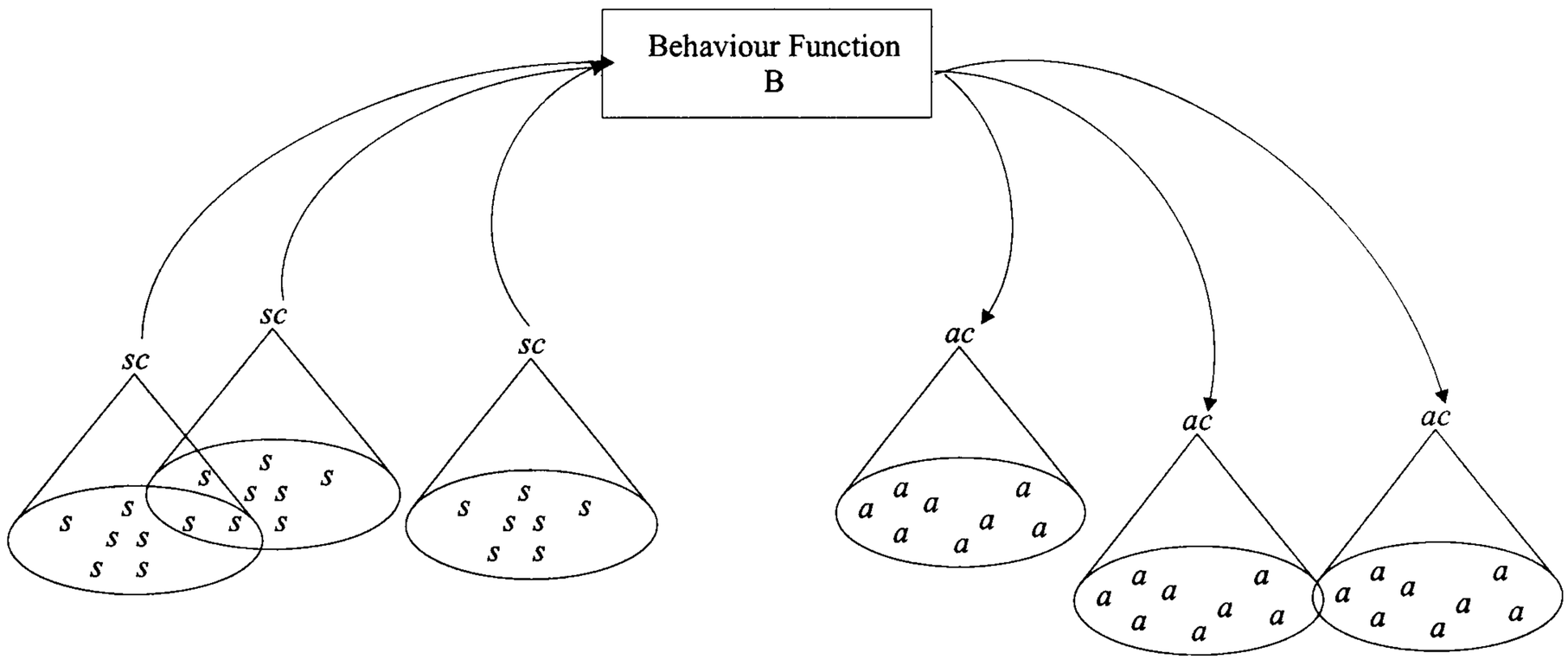


Figure 4-15: Behaviour function mapping state concepts into action concepts

Assuming that a set of state and action concepts is available (see Section 4.4), a behaviour function B, must be defined that maps state concepts into action concepts as illustrated in Figure 4-15. For any state concept sc, the function indicates the probability of selecting an action concept ac. Assuming that some examples of the desired behaviour are also available, in the form of atomic state and atomic action pairs (s, a), the behaviour function B, can be learned as follows:

1. Observe an atomic state s, from the examples.

2. Use the hypothesis defined over states to classify s into its corresponding state concept sc.

3. Observe the atomic action a, related to s in the examples.

4. Use the hypothesis to classify a, into its corresponding action concept ac.

5. Update the probability of selection ac, given sc by:

$$P(ac|sc) = \#(ac \wedge sc)/\#sc$$

   where # indicates the number of elements.

6. Repeat the process for all of the given examples.

4.5.2 Robot control

A robot using an architecture defined over atomic states and atomic actions can be controlled using a function that maps states into actions. For example, an RL robot can be controlled using a policy. That is, at every time-step

the robot perceives a state. For this, the policy indicates which is the action that should be selected so that the reward is maximised. This action is then sent to the robot's actuators.

In the architecture presented here, a similar approach is taken. That is, the previous behaviour function is now used to map state concepts into the action concept to execute. Robot control is achieved as follows:

1. Observe the atomic state s, from the environment using the robot's sensors.

2. Use the hypothesis defined over states to classify s, into its corresponding state concept sc.

3. Map sc, into the behaviour function B, and select with the probabilities indicated an action concept ac.

4. Select the representative or representatives of ac and send them to the actuators.

5. Repeat the process.

This action selection strategy resembles that of a reactive architecture (see Section 2.2.2) where actions are selected based on the stimulus received from the sensors.

4.6 Summary

This chapter has introduced a novel architecture for behaviour learning and robotic control. The architecture presented is based on the definition of states

and actions. The states describe the environment as perceived by the robot's sensors. The actions refer to the combination of motor commands available to the robot.

Motivated by the difficulty of learning complex tasks from low-level state and action representations, the architecture introduces the idea of exploiting intermediate representations, i.e. state and action concepts, based on a hierarchical lattice aggregation of their low-level representations.

The concepts generated can be of two types, concepts with a null relation between their primitives as generalisation concepts, and concepts with non-null relations as relational concepts. Generalisation concepts simply aggregate primitives into more compact representations. Relational concepts create new relational structures that have emergent properties.

The architecture learns behaviours in two steps, the first related to concept generation and second related to behaviour learning. The concept generation component classifies atomic states and actions into state and action concepts. Depending on the methods used for this classification, generalisation and relational concepts can be defined. The behaviour learning and control component uses state and action concepts to learn a probabilistic function known as a behaviour function. Learning the probabilities of the behaviour function is achieved by observing examples of the desired behaviour.

Finally, robot control is achieved by observing the current state and using the behaviour function to select an action probabilistically.

Chapter 5

Experimental Results: Generalisation Concepts in Low-Level Behaviour Learning

This chapter provides an experimental analysis of the architecture presented in the previous chapter. The experiments are based on learning a simple navigation task. The aim of this chapter is to demonstrate how generalisation concepts can be acquired from robotic sensor and motor data, and to show how these concepts can be used for behaviour learning and robot control.

More precisely, a robot is pre-programmed with a hand-coded navigation behaviour, which is used to acquire data about its interaction with the environment. The data consists of the states that the robot perceives and the actions the hand-coded behaviour triggers. This data is then analysed and generalisation concepts for this task are generated. The resulting concepts are used for learning the navigation behaviour, by using some of the exam-

ples from the hand-coded robot navigation. Tests on the accuracy of the hand-coded and the learned behaviour are compared.

5.1 Experimental test-bed

The navigation task is based on the RoboCup small-size league test-bed. The test-bed for this experiment comprised a mobile robot, a vision system and a target object. The mobile robot and the target object are colour labelled, so that the vision system is capable of tracking their position. The vision system provides positional information at 30 Hertz. Figure 5-1 illustrates the test-bed, where rx and ry are the robot's position coordinates, and $r\theta$ is its orientation; tx and ty are the target's position coordinates. These values are obtained from the vision system's coordinates. The vision system has a resolution of 640×380 pixels.

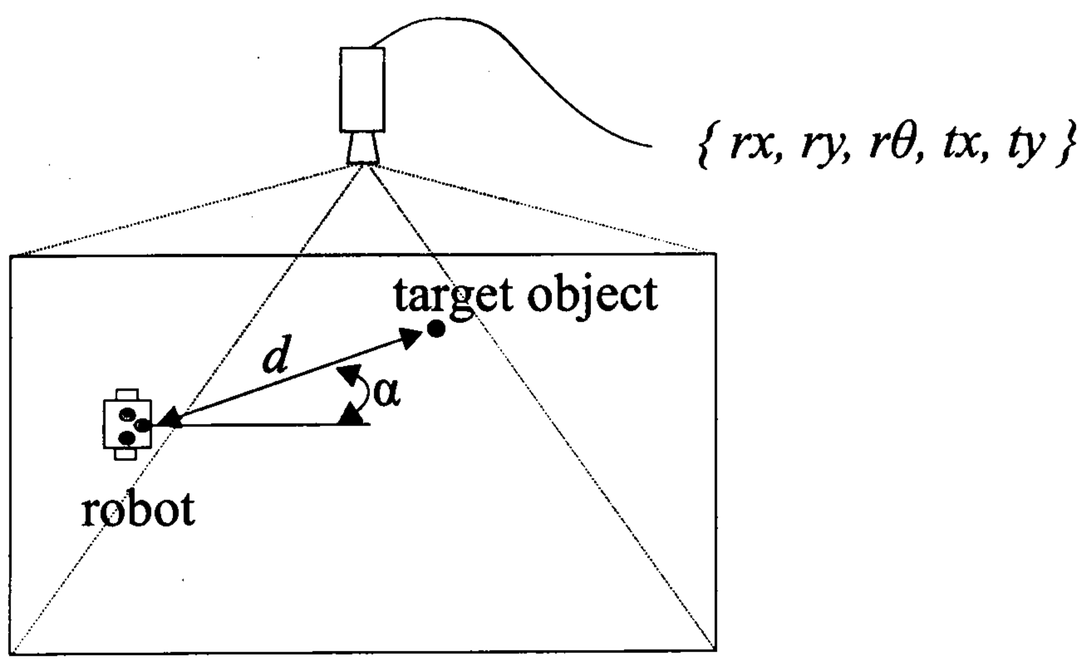


Figure 5-1: Test-bed

The behaviour to be learned is one that navigates the robot from its initial position towards the target object (a golf-ball in the physical test-bed). In this simple example, there are no obstacles involved.

The robot's state space is formed by the distance d, and the angle α to the target position (see Figure 5-1). These can be easily calculated from the vision system's data. Given the vision's resolution of 1cm, and the greatest distance between the robot and the target (the diagonal of the vision area), which is 120cm, then the distance potentially has 120 different values. The angle value ranges between 0° and 360°. In total, the robot's state space is formed by $120 \times 360 > 432 \times 10^2$ atomic states. The robot has two motors, one for each wheel. Each motor can be activated by a value from 0 to 100% of the total electric power. Thus, the action space has $100 \times 100 = 10^4$ possible atomic actions. In total the state-action space is 432×10^6.

As this state-action space is very large, we need to reduce its size. To this end, the following section describes how states and actions can be classified into generalisation concepts as we reported in [Iravani *et al.*, 2004].

5.2 Generalisation state and action concepts

One of the steps for behaviour learning by the architecture is to generate a set of state concepts and action concepts by classifying state and action primitives. As seen in Section 3.1.2, two types of concepts can be defined according to whether the relation between their primitives is $R = null$ or $R \neq null$. This chapter concentrates on concepts with $R = null$, i.e. generalisation concepts. In the experiment, the data acquired by the hand-coded robot behaviour is treated as the primitives from which to generate concepts.

To decide an appropriate classification technique to generate concepts, the following domain characteristics are taken into account:

- Nature of the data. What kinds of variables are used to define each

primitive (numeric, discrete, ordinal, binary or combinations)?

- Supervised/Unspervised learning. Is the primitive labelled with the concept it belongs to?
- Relation among primitives. Are the relations among primitives necessary?

Sensor data from the vision system are represented by interval variables (Section 3.1.3), i.e. distance d from 0 to 120 and an angle α from 0 to 360. These values are characterised by random fluctuations due mainly to noise in the vision system. The motor data is also represented by interval variables, i.e. each motor can be activated from 0 to 100% of the total power. The data acquired by the hand-coded robot is defined by the state observed and the action performed. Thus the primitive states are not labelled with their corresponding state concepts, nor are the action primitives. Generalisation concepts do not require any relation among their primitives; they only require primitives to be aggregated into a compact concept.

These characteristics require the classification methods to be based on unsupervised learning, which is capable of dealing with interval variables and does not require any relations among primitives. Therefore, distance based clustering techniques (Section 3.2.2), such as *K-means* [Duda and Hart, 1973], are good techniques for this application: they are unsupervised classification methods, they can deal with interval variables, and the classes or clusters are not defined on the basis of any relation between primitives, that is, any of the primitives belonging to the concept are sufficient.

K-means generates k exclusive concepts $c_1, c_2, ..., c_k$ where the representative of each cluster c_i is defined by the mean of the elements it contains.

K-means is an off-line clustering algorithm that defines clusters using a dissimilarity metric based on Euclidean distance. That is, each primitive to cluster is represented by a point in a multidimensional coordinate space; points that are near are considered more similar than points that are far apart.

Algorithm 1 illustrates the steps of the K-means clustering technique. The parameter k indicates the number of desired clusters, and the weights in W are used to adjust the relevance of each of the p variables that describe the p-dimensional data. The dissimilarity is measured by the Weighted Euclidean Distance between each data point and each cluster centre. The cluster that is the most similar to the data point is considered to include the data point. When data points are added to a cluster its centre is updated towards the mean value of all the data points included. The algorithm takes as input: the number of desired clusters k, the data-set to cluster X, and a set of weights w. Each element $x \in X$ is a p-dimensional point. The algorithm outputs the resulting position of the cluster centres after the clustering.

The algorithm operates as follows. In (1) each of the cluster centres is randomly initialised; in (2) the position of the centres are stored; in (3) the distance d_j between data point i and cluster centre j is measured; in (4) the cluster j with minimum distance from the data-point i is stored in m_i; in (5) each cluster centre is updated towards the mean value of all the data-points found to be closest to it. This process is repeated until the update function stops changing the cluster centres, i.e. $c = c'$.

Algorithm 1: K-means

Input: k, number of clusters;

X, set of data to cluster, where each element, $x \in X$, is p-dimensional;

$w = \{w_1, w_2, ..., w_p\}$, a set of p weights;

Output: $C = \{c_1, c_2, ..., c_k\}$, set of k cluster centres. Where each centre is a p-dimensional point;

for $c_i \in C$ **do**

1 randomly initialise c_i;

end

2 $C' \leftarrow C$;

repeat

for $x_i \in X$ **do**

3 **for** $c_j \in C$ **do**

$d_j = \sqrt{w(x_i - c_j)^2}$;

end

4 **for** $d \in d_j$ **do**

$m_i = \min_j(d_j)$;

end

end

5 C = update (C, X, m);

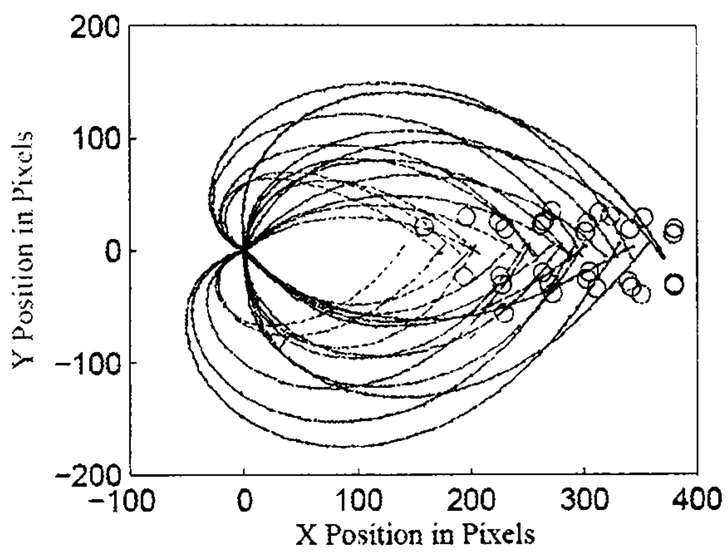
until $C = C'$;

The following section illustrates the resulting concepts after applying the K-means clustering algorithm to the data acquired by the hand-coded robot.

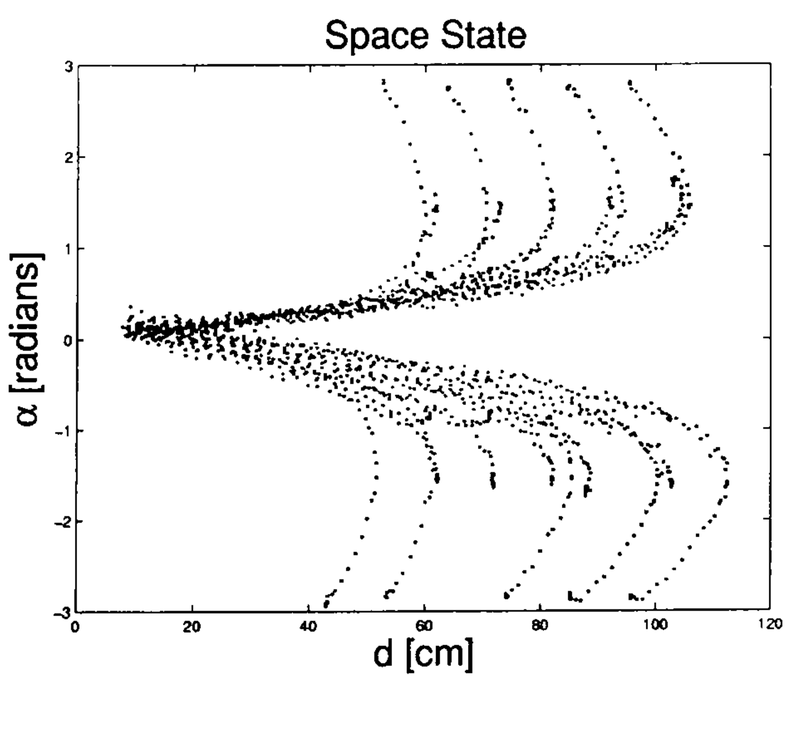
5.2.1 Clustering states and actions

The hand-coded robot behaviour is used to acquire data from the navigation task. In this experiment 30 paths were generated from different initial positions to a target object. Figure 5.2(a) illustrates the 30 paths; for presentation purposes, these have been drawn from the same initial position.

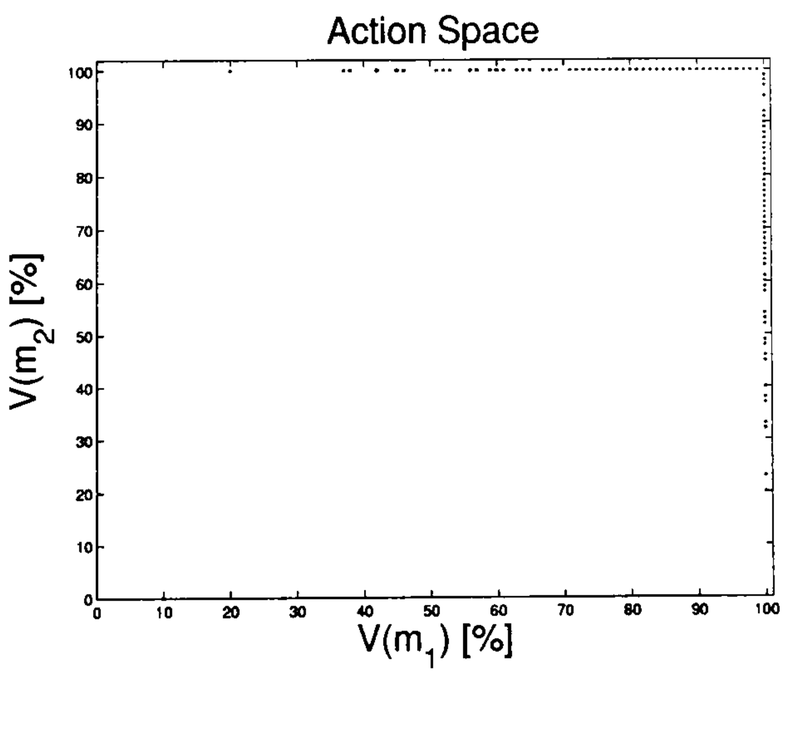
During the execution of these paths, the robot recorded the states encountered and the actions selected. Figure 5.2(b) illustrates the atomic states perceived, where the distance variable d is represented in centimetres and the angle variable α is represented in radians. Figure 5.2(c) illustrates the atomic actions executed, where the motor activations $V(m_1)$ and $V(m_2)$, are represented as the percentage of the total power applicable. The last figure illustrates that the hand-coded behaviour always sets the power of one of the wheels at 100% and varies the other. This can be seen by the actions which always follow the lines of $V(m_1) = 100$ or $V(m_2) = 100$.



(a) Hand-coded robot's paths



(b) State space



(c) Action space

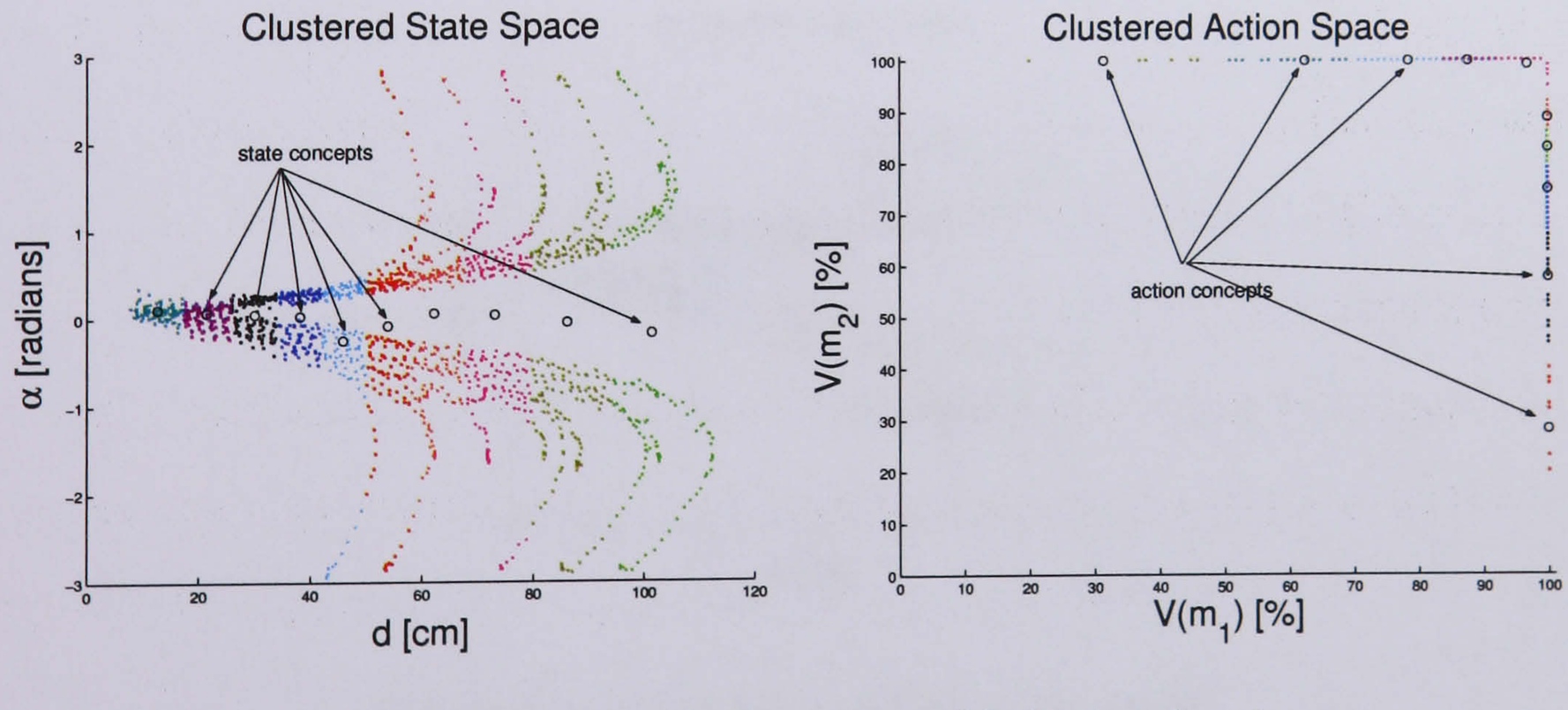
Figure 5-2: Paths, perceived states and executed actions

The K-means algorithm has been applied to the data acquired by the robot, using the parameters in Table 5.1, where $\{w_{V(m_1)}, w_{V(m_2)}\}$ are the weight values related to the two-dimensional action space; and $\{w_d, w_\alpha\}$ are the weight values related to the two-dimensional state space. In the experiment the parameter $k = 10$ was selected arbitrarily as the experiment is only intended to demonstrate the shortcomings of this clustering technique. To address this shortcoming, Section 5.6 presents a novel k-means classification technique which does not require the *a priori* definition of k.

Table 5.1: K-means parameters

	action clustering	state clustering
k	10	10
weights	$w_{V(m_1)} = 1$ $w_{V(m_2)} = 1$	$w_d = 1$ $w_\alpha = 1$

The results of clustering the previous atomic state and actions are illustrated in Figure 5-3.



(a) Clustered state space (b) Clustered action space

Figure 5-3: Clustered state and action spaces

Figure 5.3(a) illustrates the resulting state concepts while Figure 5.3(b) illustrates the resulting action concepts. Each concept is represented by a different colour and the representative of the cluster is shown by a circle. The parameters in Table 5.1 indicate that the number of clusters or concepts is 10 for both spaces, and that all weights are set to 1. These weights imply that the Euclidean distance (dissimilarity) between data points is measured based on the original units in which the data was given. That is, as Figure 5.3(a) illustrates, the state concepts defined when the distance d is measured in centimetres and the angle α is measured in radians.

As the range of the angle variable (i.e. 2π) is smaller than the distance variable (i.e. 120), the distance variable will have predominance over the angle. In other words, the distance variable will account for most of the dissimilarity in the state space. This effect is observable in Figure 5.3(a) in which all clusters are mostly aligned with the distance axis. Thus, clusters are independent of the value of the angle.

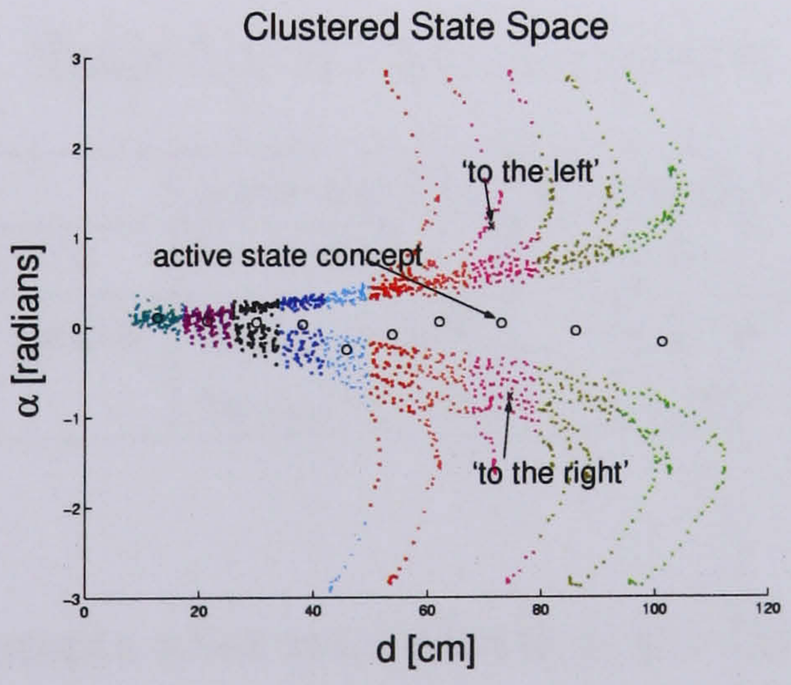


Figure 5-4: Concept indifferent to angle

This scaling problem will have negative effects when trying to control the

robot using concepts. For example, if two different state are indicating that the robot is located (i) to 'the right' or (ii) 'to the left' with respect to the target, the state concept related to the atomic states will represent them indifferently; this is shown in Figure 5-4. A robot using these state concepts would not possess the concept representing that: 'target is to the right' or 'target is to the left'. A robot that can't perceive these two states as different will not be able to turn in the corresponding direction, thus failing to reach the target position.

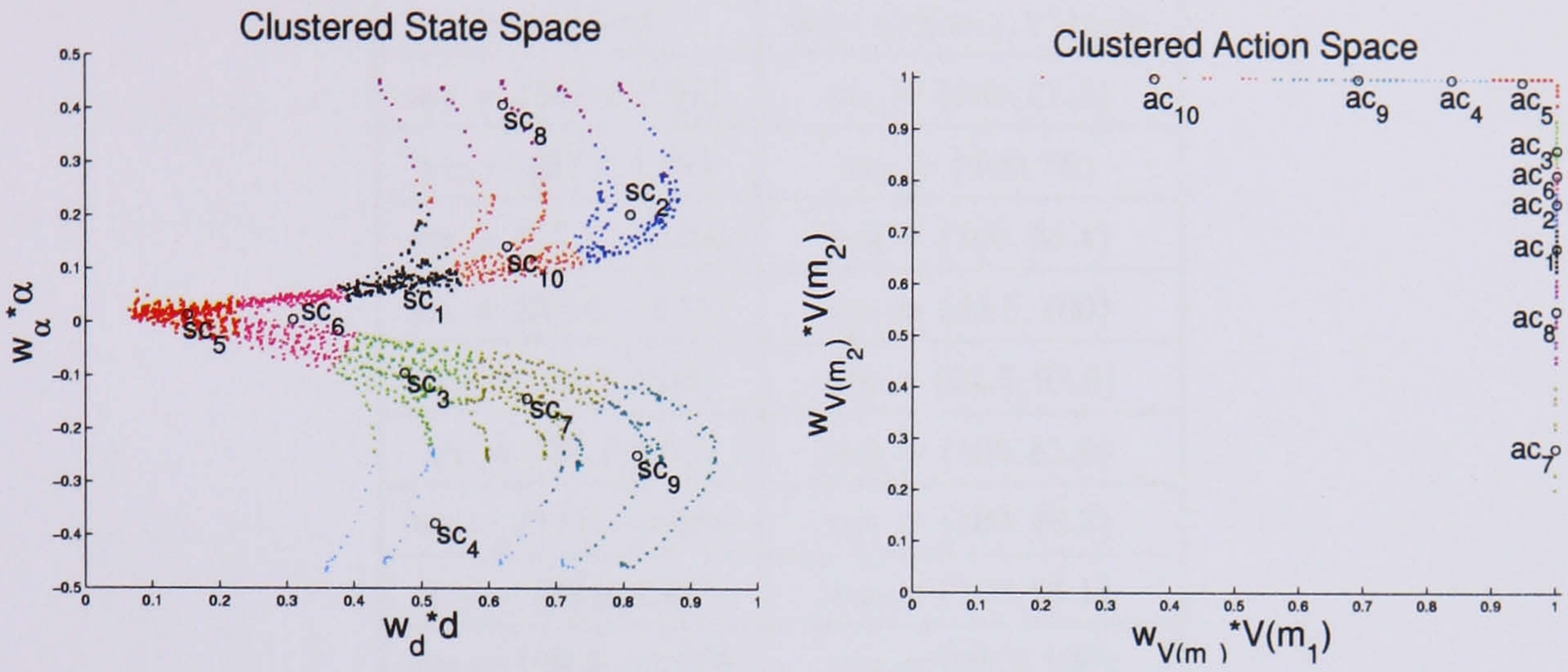
This problem can be solved by adjusting the weight values associated with each variables. Finding the correct value for the weights relates to the *chalk and cheese* problem described in Section 3.3.1. To alleviate the scaling problem, *normalisation* approaches can be used, where the range of all variables is normalised to a 0 to 1 range, thus forcing all variables to have the same relevance towards the definition of dissimilarity. The weight parameters in Table 5.2 are used to normalise the previous variables.

Table 5.2: K-means parameters

	action clustering	state clustering
k	10	10
weights	$w_{V(m_1)} = 1/100$ $w_{V(m_2)} = 1/100$	$w_d = 1/120$ $w_\alpha = 1/2\pi$

The resulting concepts after normalisation are illustrated in Figure 5-5. As it can be seen in Figure 5.5(a) the state concepts are now influenced by both the distance and the angle variables, i.e. state concepts are no longer aligned with the distance variable. The robot is now able to recognise target to the right or target to the left states. For example, the same atomic states

used previously: 'to the right' and 'to the left', now activate two different concepts rather than just one (see Figure 5-6).



(a) Weighted state space

(b) Weighted action space

Figure 5-5: Clustered weighted state and action spaces

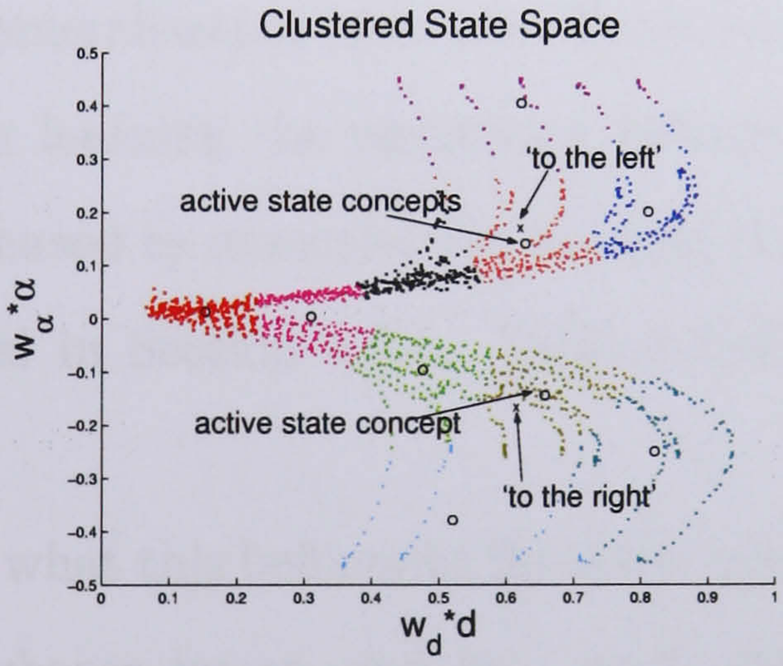


Figure 5-6: Concept dependent on distance and angle.

Table 5.3 shows the unscaled (i.e. without being multiplied by the weights) position of the representatives for the previous state and action concepts.

Table 5.3: Concept representatives

state $sc = \{d, \alpha\}$	action $ac = \{V(m_1), V(m_2)\}$
$sc_1 = \{56.34, 0.51\}$	$ac_1 = \{100, 67.3\}$
$sc_2 = \{97.3, 1.27\}$	$ac_2 = \{100, 76\}$
$sc_3 = \{57.1, -0.60\}$	$ac_3 = \{100, 86.4\}$
$sc_4 = \{62.4, -2.38\}$	$ac_4 = \{83.5, 100\}$
$sc_5 = \{18.4, 0.08\}$	$ac_5 = \{94.5, 99.6\}$
$sc_6 = \{37.2, 0.03\}$	$ac_6 = \{100, 81.6\}$
$sc_7 = \{78.8, -0.90\}$	$ac_7 = \{100, 28.2\}$
$sc_8 = \{74.6, 2.55\}$	$ac_8 = \{100, 55.1\}$
$sc_9 = \{98.4, -1.57\}$	$ac_9 = \{68.9, 100\}$
$sc_{10} = \{75.3, 0.89\}$	$ac_{10} = \{37.3, 100\}$

5.3 Learning the navigation behaviour

Given the previous generalisation concepts (Table 5.3), these are now used as representations for learning the navigation behaviour. The paths previously used are now reused as examples for learning the behaviour, following the method presented in Section 4.5.1. Table 5.4 illustrates the resulting behaviour function.

As an example of what this behaviour function represents, let us compare the action concepts chosen by sc_1 and sc_4. sc_1 is related with a relatively high probability to ac_6 (0.51%), while sc_4 is highly related to ac_{10} (81%). Table 5.3 gives the information for each of these concepts' representatives. That is $sc_1 = \{56.34, 0.51\}$, $sc_4 = \{62.4, -2.38\}$, $ac_6 = \{100, 81.6\}$ and $ac_{10} = \{37.3, 100\}$. As we observe from the sc_1 and sc_4 representatives, both have similar distances $(56.34 : 120)$ and $(62.4 : 120)$ but dissimilar angles $(0.51 : \pi)$ and $(-2.38 : -\pi)$. In other words, sc_1 and sc_4 'mean' that the

robot is at a similar distance from the target, but 'slightly to the left' and 'largely to the right', respectively. It is logical to think that, if the robot is observing sc_1 it must 'turn slightly to the left', whereas if observing sc_4 it must 'turn hard to the right', and this is exactly what action concepts ac_6 and ac_{10} do (see their motor velocities' relations).

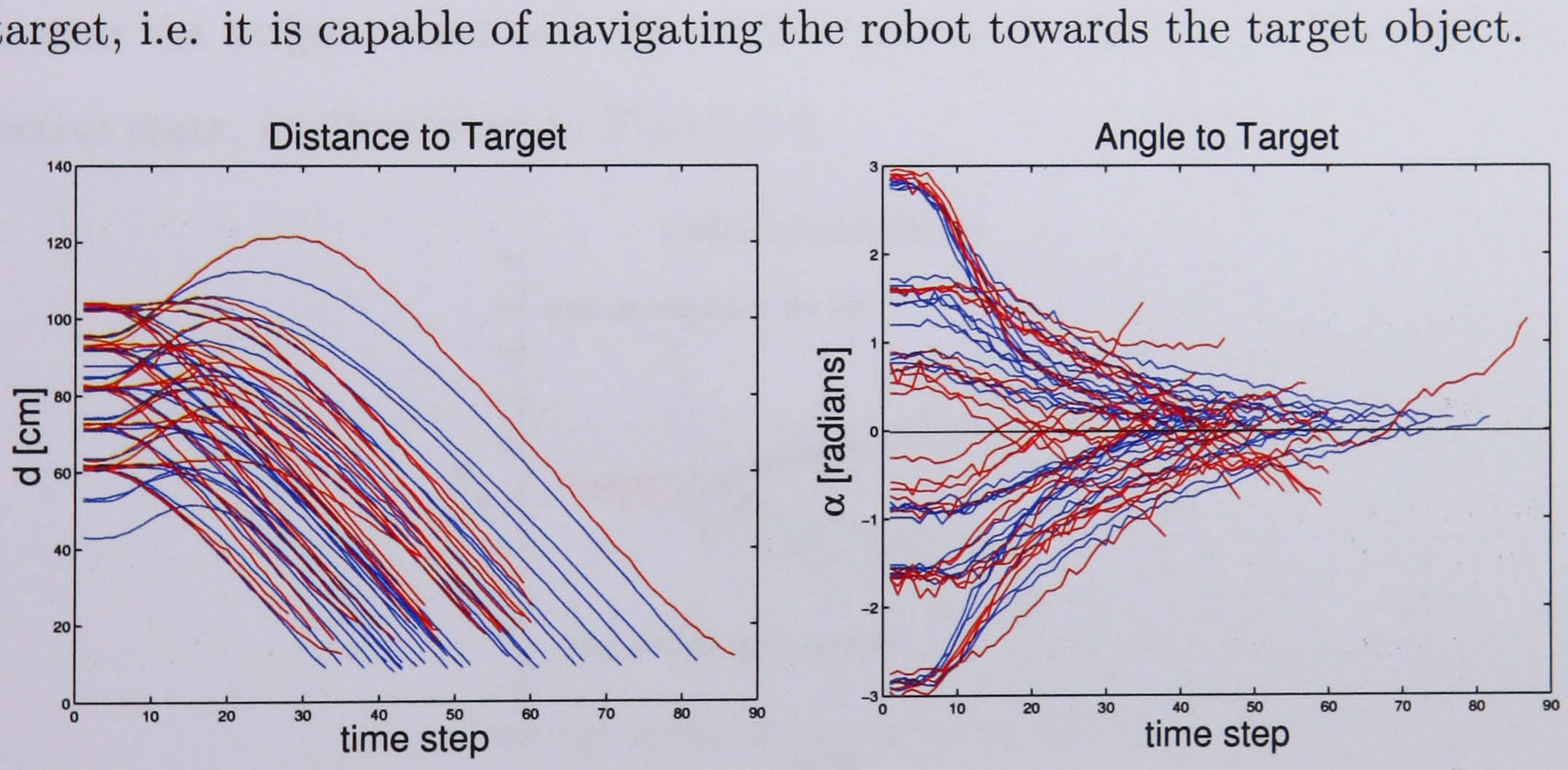
Table 5.4: Behaviour function

	ac_1	ac_2	ac_3	ac_4	ac_5	ac_6	ac_7	ac_8	ac_9	ac_{10}
sc_1	0.02	0.12	0.29	0	0	0.51	0	0.06	0	0
sc_2	0.32	0.31	0	0	0	0.26	0	0.10	0	0
sc_3	0	0	0	0.53	0.21	0	0	0	0.24	0.01
sc_4	0	0	0	0	0	0	0	0	0.19	0.81
sc_5	0.04	0.09	0.39	0.06	0.28	0.14	0	0	0	0
sc_6	0	0	0.31	0.24	0.27	0.19	0	0	0	0
sc_7	0	0	0	0.72	0.06	0	0	0	0.22	0
sc_8	0	0	0	0	0	0	0.88	0.12	0	0
sc_9	0	0	0	0.37	0	0	0	0	0.52	0.11
sc_{10}	0.1	0.18	0.13	0	0	0.40	0.01	0.19	0	0

5.4 Navigation behaviour for control

Once the probabilities of the behaviour function have been learned, it can be used to control a robot. As seen in Section 4.5.2, the behaviour function is used to control the agent as follows: by (i) observing the active state of the environment s_t; (ii) relating s_t to the concept state sc_t. This is achieved by locating the cluster (state concept) closest to s_t; (iii) selecting an action concept with the probabilities indicated by the behaviour function; and, finally (iv) sending the representative of the action concept selected to the motors.

Figure 5-7 illustrates the distance and the angle variables of the robot while it navigates towards the target position. The target position is reached accurately if the distance and angle at the end of the trajectory are 10 cm and 0 radians, respectively. Figure 5-7 represents in *blue* the trajectories of the robot when controlled using the hand-coded program, and in *red* the trajectories when controlled by the learned behaviour function. Figure 5.7(a) illustrates the distance variable, as it can be observed: the hand-coded (blue) robot outperforms the learned behaviour (red), as the robot's end position approximates better to the desired value. Figure 5.7(b) illustrates the angle variable; again, the hand-coded robot provides a better convergence towards the target of 0 radians. Despite the loss in navigation accuracy, the behaviour function is capable of reducing the initial distance and angle towards the target, i.e. it is capable of navigating the robot towards the target object.



(a) Distance from target

(b) Deviation from target

Figure 5-7: Hand coded vs concept behaviour control

Although these results may look discouraging at first, the following issues should be taken into account:

- The navigation behaviour is being represented compactly, by using only 10 state concepts and 10 action concepts, thus $10 \times 10 = 100$ table positions are necessary to store this information, compared to the 432×10^6, when using the original state and action spaces.

- The behaviour function drives the robot accurately during the initial parts of the trajectory, but starts losing accuracy when approaching the target.

This last observation is illustrated in Figures 5.7(a) and 5.7(b), where it can be seen that the distance and angle variables are similar for both controllers in the initial stages of the trajectory; but diverge when the angle approaches 0 radians. The reason for this divergence, is that when the robot is near the target object and the angle is around 0 radians, sc_6 becomes the active state, as illustrated in Figure 5-8.

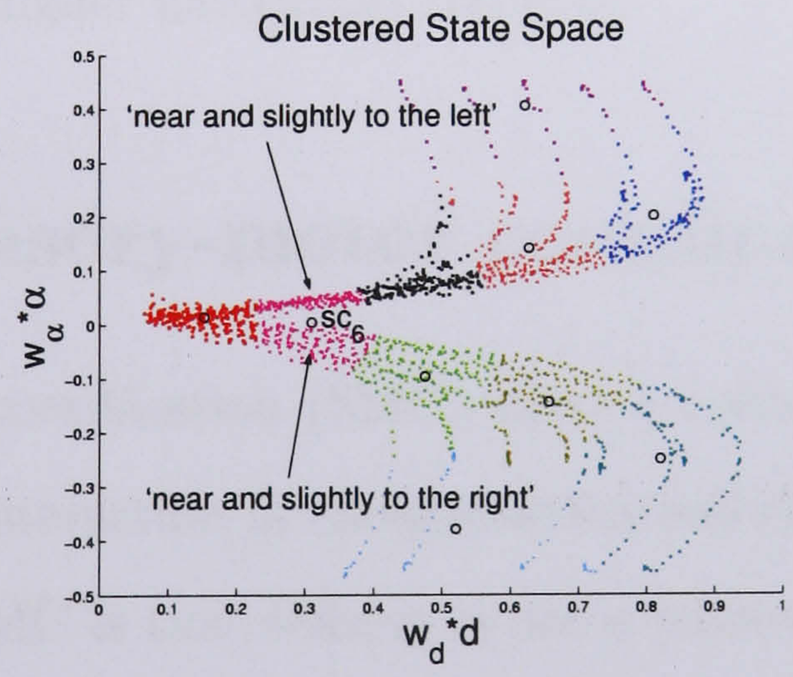


Figure 5-8: Active state concept when robot is 'near' the target

Being 'near and slightly to the left' or 'near and slightly to the right' of the target are part of the same state concept sc_6, thus it is not possible to

differentiate two states which require different control actions (turning right and turning left).

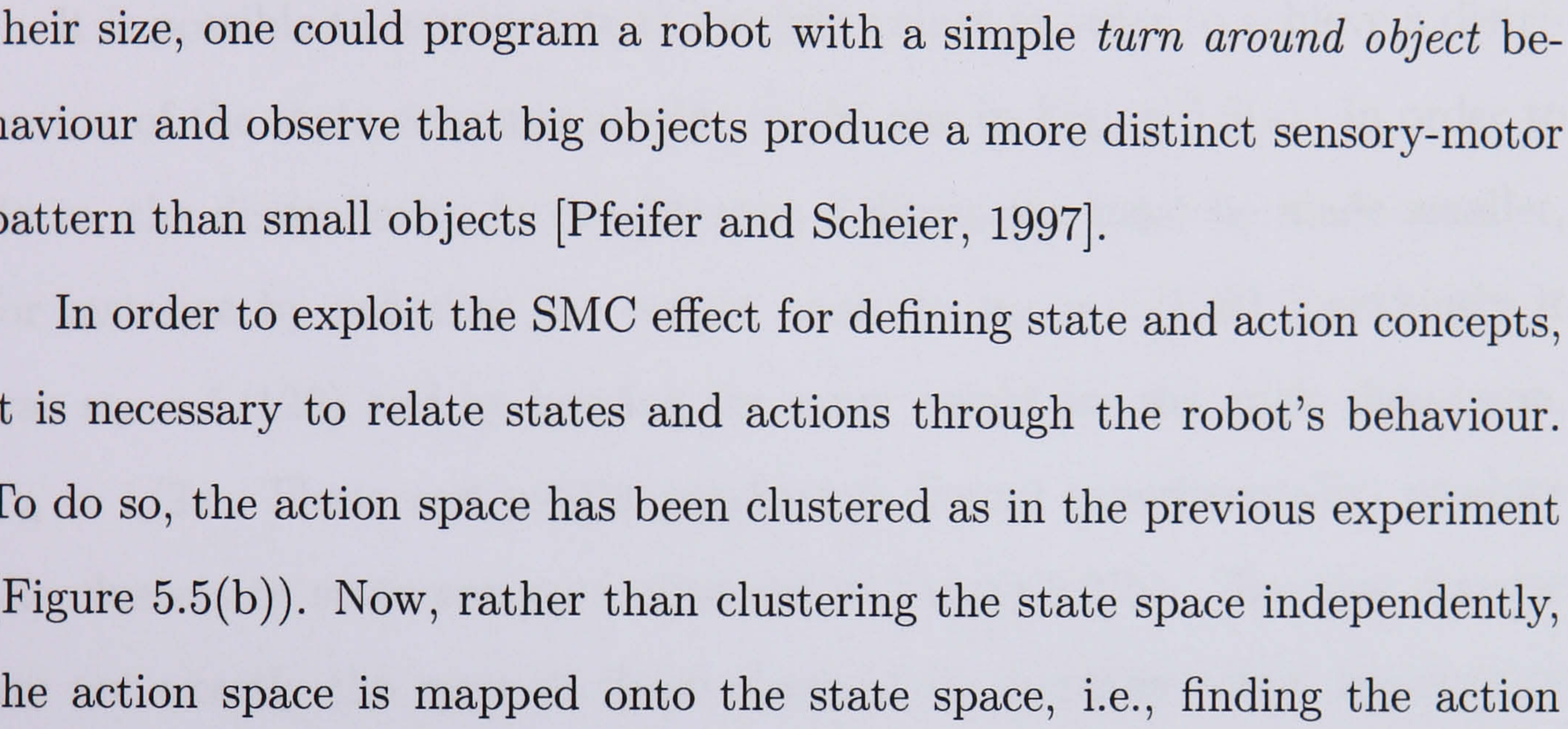
The behaviour function (Table 5.4) shows that once in state concept sc_6, the robot has high probabilities (shown in brackets) of choosing ac_3 (0.31), ac_4 (0.24) and ac_5 (0.27). $ac_3 = \{100, 86.4\}$ is a 'turn slight to the left', $ac_4 = \{83.5, 100\}$ is a 'turn slight to the right' and $ac_5 = \{94.5, 99.6\}$ is 'forwards'. These probabilities indicate that when the robot is in state sc_6 it faces the indecision of whether to turn slightly to the left, slightly to the right or going forward. This problem is due to sc_6 being too general and not being able to categorise as different classes the 'near and slightly to the left' and 'near and slightly to the right' situations. This problem exemplifies the importance of finding a correct representation of state and action concepts. The following section shows how the *sensory-motor coordination effect* can be used to create concept representations for the navigation task that is designed to achieve higher navigation accuracy.

5.5 The sensory-motor coordination effect

The sensory-motor coordination (SMC) effect emphasises the usage of the robot-environment interaction in categorisation and classification tasks. An important idea in SMC is that sensing is not a passive function carried out by the robot's sensors, but it is an active function that requires both sensors and motors to interact in the environment in order to discover classes [Pfeifer and Scheier, 1997, Pfeifer and Scheier, 2001]. SMC has been applied mainly to robots controlled by reactive or embodied architectures. In the context of these architectures, the coordination between sensors and motors

is the result of the interaction between the robot, the behaviours and the environment (see Section 2.2.2). The SMC effect simplifies the classification task by inducing correlations or patterns between the sensory-motor spaces and the classes. For example, in order to discriminate objects on the basis of their size, one could program a robot with a simple *turn around object* behaviour and observe that big objects produce a more distinct sensory-motor pattern than small objects [Pfeifer and Scheier, 1997].

In order to exploit the SMC effect for defining state and action concepts, it is necessary to relate states and actions through the robot's behaviour. To do so, the action space has been clustered as in the previous experiment (Figure 5.5(b)). Now, rather than clustering the state space independently, the action space is mapped onto the state space, i.e., finding the action concepts used at each state (SMC space).



(a) Sensory-motor space

(b) State concepts after SMC

Figure 5-9: Sensory-motor space and resulting state concepts

Figure 5.9(a) illustrates the mapping of actions concepts into states, where each colour corresponds to the different action concepts. The resulting

distribution shows that action concepts mainly vary along the angle α dimension, while stay constant along the distance d dimension. This distribution indicates that changes in the angle dimension result in more dissimilar states than changes in the distance dimension.

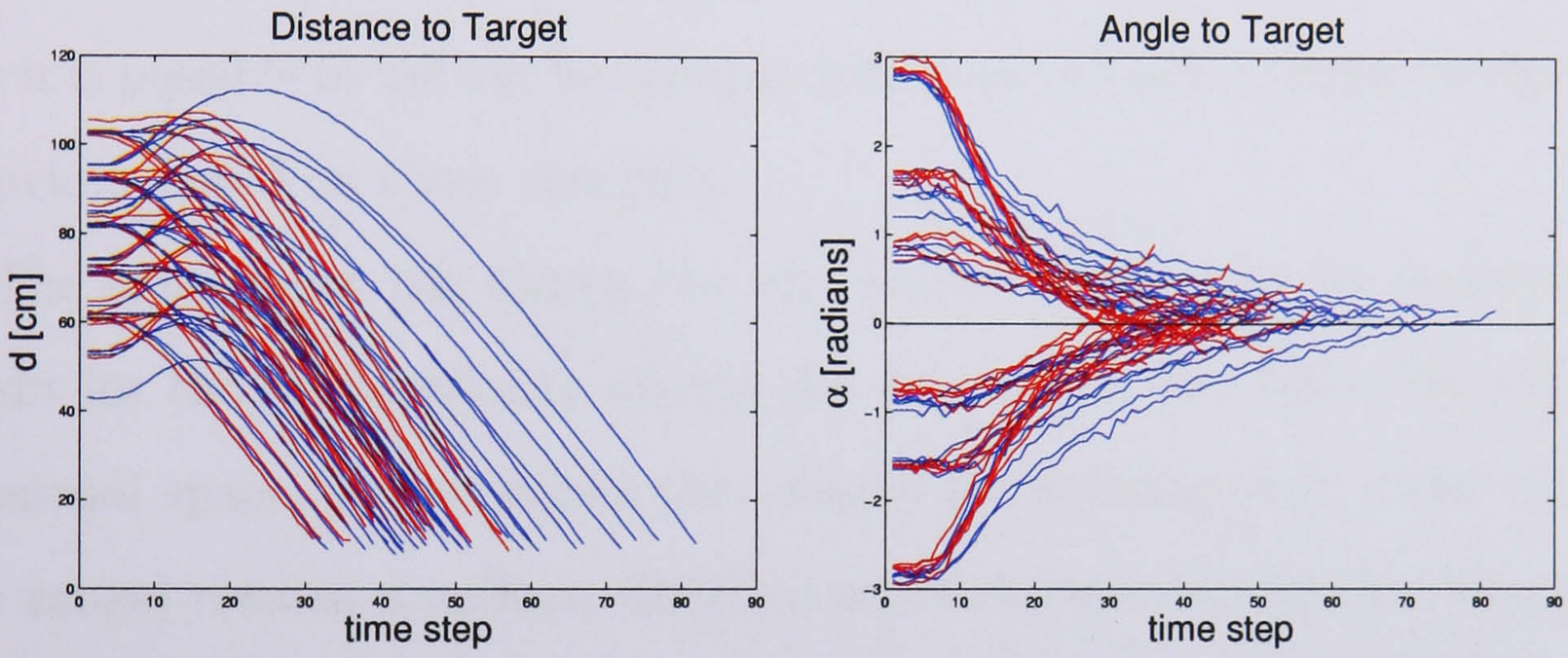
It is possible to manipulate the weight values in order to achieve a distribution of the state concepts similar to the one in Figure 5.9(a). In order to do so, the dissimilarity in the distance d dimension must be made smaller, for instance by reducing the weight value to $w_d = 1/1080$ (previously it was $w_d = 1/120$) and by keeping the same weight for the angle dimension, $w_\alpha = 1/2\pi$. These new weight parameters (found experimentally) produce the clusters or state concepts observed in Figure 5.9(b). The new clusters are not exactly the same as those obtained by mapping action concepts in the SMC space, but they maintain a certain similarity, i.e. state concepts vary according to the angle dimension.

Table 5.5: Behaviour function after SMC

	ac_1	ac_2	ac_3	ac_4	ac_5	ac_6	ac_7	ac_8	ac_9	ac_{10}
sc_1	0	0	0	0	0	0	0.87	0.13	0	0
sc_2	0	0.02	0.45	0	0	0.52	0	0	0	0
sc_3	0	0	0	0	0	0	0	0	0.12	0.88
sc_4	0.42	0	0	0	0	0	0.12	0.46	0	0
sc_5	0.03	0.07	0.39	0.03	0.37	0.11	0	0	0	0
sc_6	0.03	0.28	0.08	0	0	0.59	0	0.02	0	0
sc_7	0.32	0.38	0	0	0	0	0	0.30	0	0
sc_8	0	0	0	0.08	0	0	0	0	0.85	0.08
sc_9	0	0	0	0.81	0	0	0	0	0.19	0
sc_{10}	0	0	0	0.67	0.32	0	0	0	0.01	0

The behaviour function must now be re-learned for these new concepts. Table 5.5 shows the probability values of the new function. Using this new function to control the robot we obtain a slightly worse accuracy than that of the hand-coded robot, but a better accuracy when compared with the behaviour learned without using the SMC effect.

Figure 5-10 illustrates the distance and angle variables during navigation. Again, *blue* is used for the hand-coded policy and *red* for the learned behaviour.



(a) Distance towards target (b) Deviation towards target

Figure 5-10: Hand coded vs concept behaviour control after SMC

Table 5.6 shows the error of each controller in 30 trajectories with similar initial and target positions. E_d and E_α are the absolute accumulated errors with respect to the optimal distance value and to the optimal angle value. For example, the hand-coded controller accumulates 39.6 cm of error in 30 trajectories. This table indicates that the hand-coded controller outperforms any of the learned behaviours. It also shows that using SMC to define concepts affects positively the performance of the learned behaviour.

Table 5.6: Navigation accuracy

	hand-coded	learned behaviour	learned behaviour + SMC
$E_d[cm]$	39.6	187.5	75.3
$E_\alpha[radians]$	3.73	14.1	7.48

5.6 Adaptive concept definition

The previous experiment demonstrated the feasibility of clustering continuous state and action spaces into generalisation concepts. It has also shown that it is possible to use our learning architecture to learn a simple navigation behaviour based on those concepts.

The experiment has shown the importance of selecting an appropriate weight for the (dis)similarity among the dimensions that form the multidimensional space. It was shown that simply normalising their scales (e.g. 0 to 1 range) creates a uniform distribution of clusters or concepts along the axis of the multidimensional spaces. This distribution is not necessarily adequate, as was shown by the low accuracy of the first navigation behaviour. To address the issue of finding a 'good' set of weights, a method based on the SMC effect was proposed. The SMC method used interrelated state and action spaces through the robot's behaviour. Such a definition of the state-action space allowed the designer to observe the pattern of actions taken in relation to the state perceived. A set of weights were then found that produced a similar pattern state-action pattern. It was shown that the navigation behaviour based on these new concepts achieved a better accuracy.

The experiment described above has shown the applicability of the architecture. Some open issues remain, which are described in the following.

Firstly, dividing the learning approach into two separate steps implies that concept learning must be performed before the resulting concepts can be used for behaviour learning. Separating learning in this manner has the following limitations: (i) data for learning must be acquired in an off-line manner; (ii) it is not clear how much data is necessary to generate state and action concepts.

Secondly, it is clear that the number of state and action concepts will directly affect the performance of the learning system. Having defined too few concepts will affect the quality of the behaviour function; in extreme cases, the desired behaviour will not be achievable. Letting the system use too many concepts will provide no generalisation over states and actions, and thus the amount of experience and computational requirements will be similar to those required by the original spaces. In other words, the selection of an appropriate number of state and action concepts is an important issue.

Thirdly, assuming that a robot is capable of generating an appropriate number of states and actions concepts for a certain task and environment, we are focusing on the problem of how the robot can maintain a useful representation in the face of possibly changing circumstances.

These issues can be partially addressed by allowing the robot to adapt its representation as well as its behaviour. This means that, given a set of state and action concepts, agents should be capable of creating and deleting concepts as necessary. Thus, when too few concepts are defined the agent should add concepts to the representation. When there are too many concepts some should be eliminated from the representation. We define an approach capable of adapting concepts as an *adaptive concept definition.*

The approach is introduced in the following section and exemplified by the same navigation behaviour example used previously. Because the state and actions have the same characteristics as before, we maintained the same clustering technique (K-means). As will be seen, some changes to the basic K-means algorithm were made in order to achieve a concept representation with adaptive capabilities.

5.6.1 Adapting action and state concepts

To implement an adaptive state and action representation we apply a heuristic based on *specialising* concepts. Initially, the robot defines a unique state and a unique action concept covering the whole of the state and action spaces. By definition, any atomic state or action will be a primitive of these concepts. Figure 5-11 illustrates these unique concepts, where S and A are the state and action spaces, and sc_{11} and ac_{11} the only concepts.

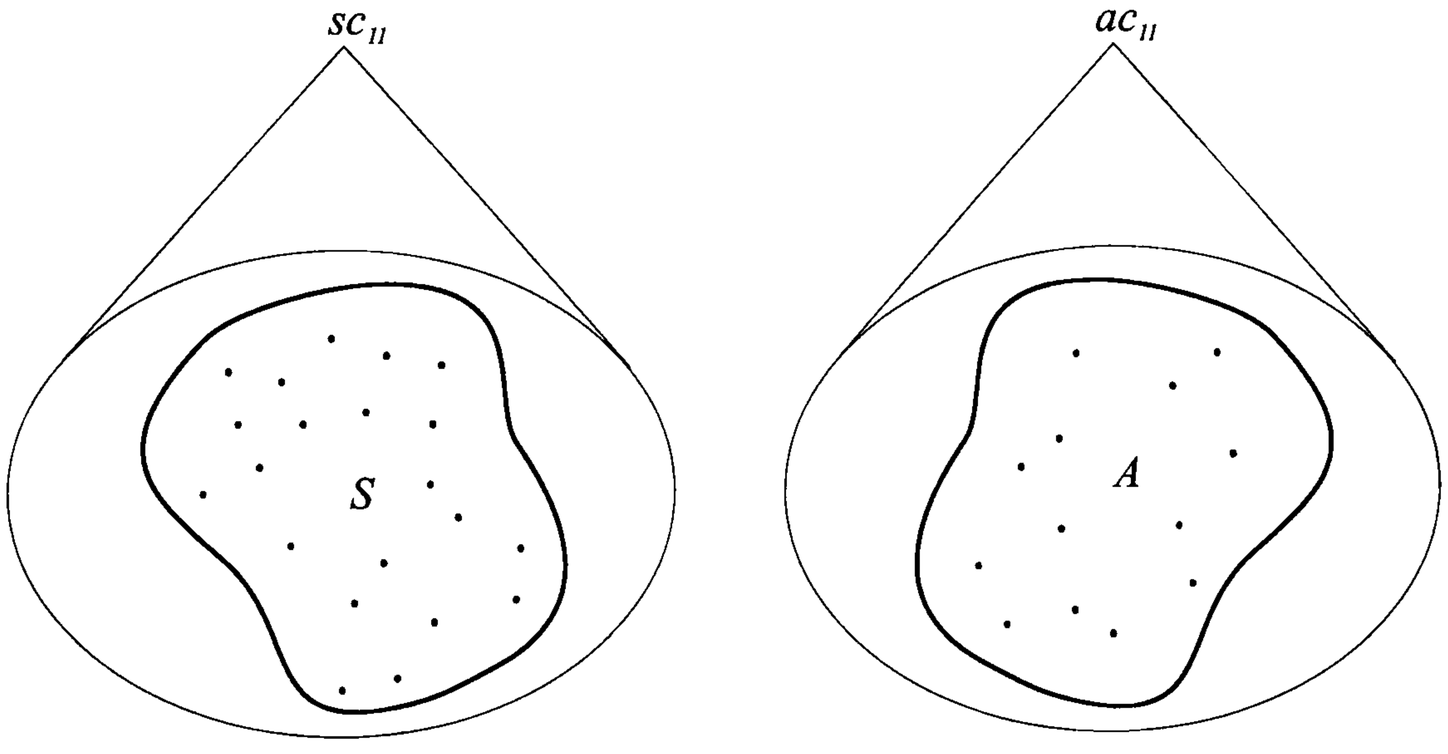


Figure 5-11: Representation with unique state and action concepts

Most complex systems need more than one state and action concept to

be represented, thus the number concepts must be increased. The approach followed in this thesis is one of specialisation by partitioning concepts.

A generalisation concept can be specialised simply by dividing the primitives it generalises into sub-concepts or *children concepts*. For example, following from the above, state concept sc_{11} and action concept ac_{11} can be partitioned into n child concepts. Here we have arbitrarily selected $n = 2$, i.e. each concept is specialised into a pair of children concepts. The result of the specialisation is illustrated in Figure 5-12, where sc_{11} is partitioned into sc_{21} and sc_{22}, and ac_{11} is partitioned into ac_{21} and ac_{22}.

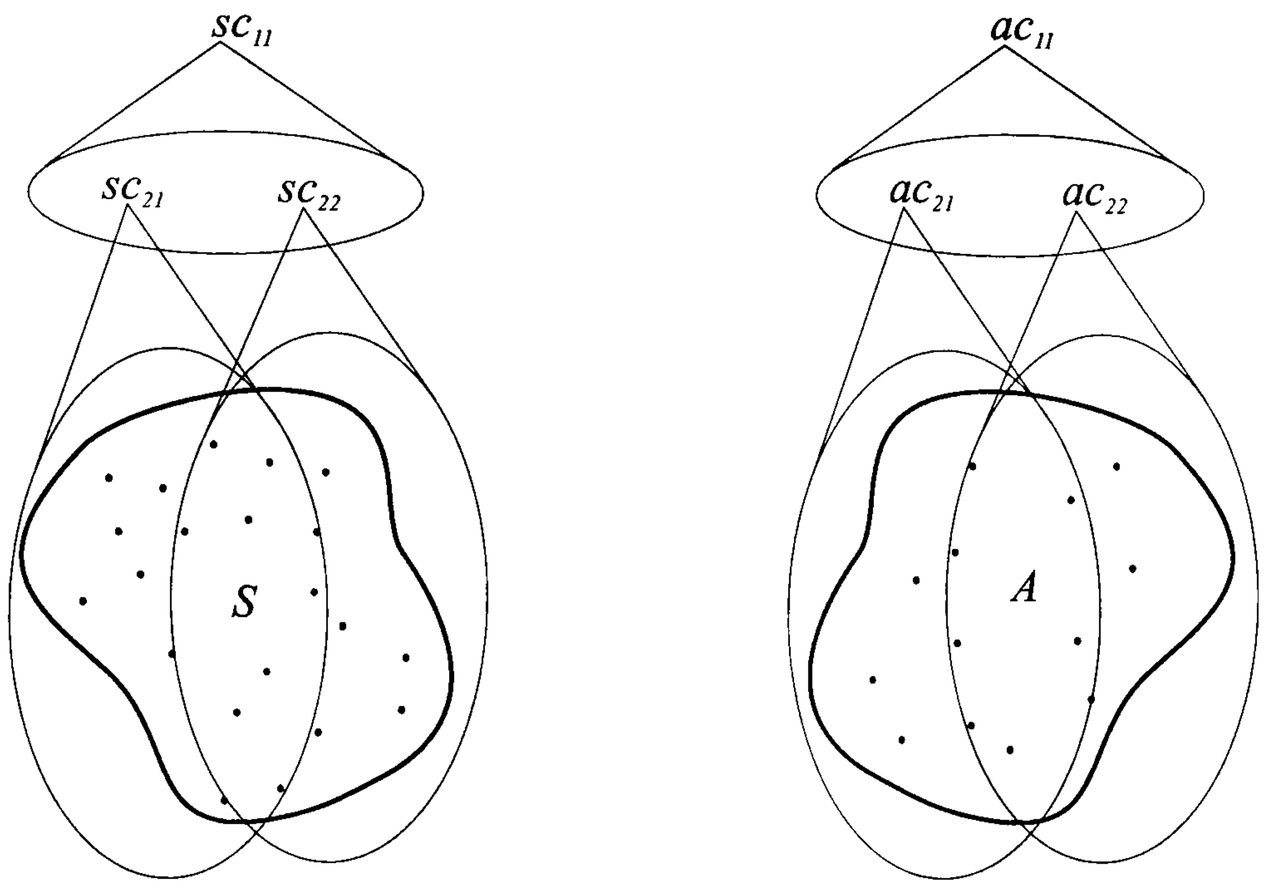


Figure 5-12: Specialised state and action concepts

The concept specialised is known as the *parent concept*, i.e. sc_{11} and ac_{11} are parent concepts. An important requirement for the children concepts is that together they must generalise the same primitives as the parent concept, so that $sc_{11} = sc_{21} \cup sc_{22}$ and $ac_{11} = ac_{21} \cup ac_{22}$. As each children concept generalises a smaller portion of primitives compared to their parents, we say

that children concepts are more specific than their parents.

These new state and action spaces can be represented in the form of a tree, where the nodes of the tree correspond to concepts and the links correspond to the relation between parent concepts and children concepts (see Figure 5-13). This example illustrates only one specialisation step, but the process can be recursively applied to all of the most specific concepts, i.e. the concept in the leaves of the tree known as *leaves concepts*, resulting in a tree-structure of concepts which can be adapted by specialising existing concepts.

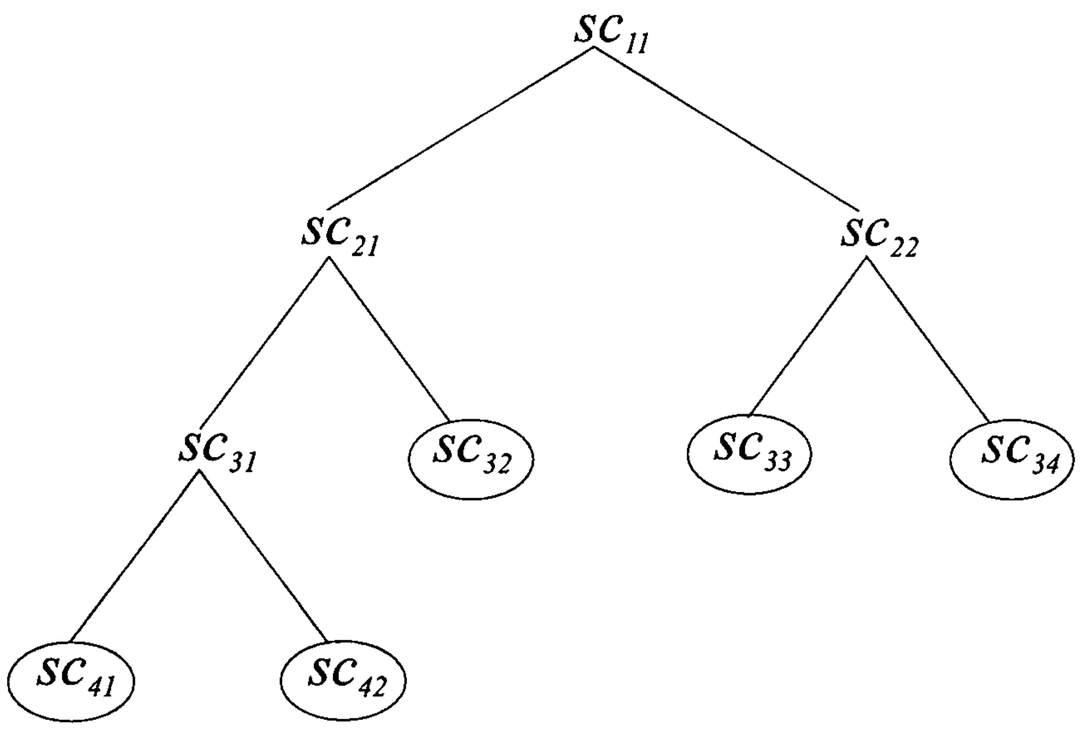


Figure 5-13: A tree of state concepts

These new types of concept are generated by modifying the previous clustering technique to deal with the notion of adaptive concepts or clusters. To do so, we have modified the K-means algorithm into an *incremental* and *tree-structured* K-means algorithm (ITS-K-means) [Iravani, 2004].

There are two main ideas behind the ITS-K-mean algorithm: (i) ITS-K-means applies the K-means algorithm (see Algorithm 1) incrementally, that is by clustering each primitive as observed by the robot, and (ii) each concept

in the tree-structure is represented by a cluster.

Applying K-means in an incremental manner allows the robot to generate concepts and adapt the representation as it interacts with the environment, thus there is no need to collect data in an off-line manner. Moreover, it makes it possible to interleave the two learning sub-tasks, i.e. behaviours are learned at the same as concepts are generated. The following section gives the details of the ITS-K-means algorithm.

5.6.2 Incremental tree-structured K-means

ITS-K-means extends the standard K-means algorithm (see Algorithm 1) by making clustering incremental, and by structuring clusters in a tree-structure. The structure is ordered by generality, that is, concepts near the root of the tree represent general concepts, whereas concepts near the leaves represent more specialised concepts.

K-means has been extended to a version that incorporates incremental clustering, known as *Adaptive K-means* [Darken and Moody, 1990]. This method clusters incoming data incrementally, in relation to previously experienced data points. In contrast to K-means, which operates on all the primitives and finds a local minimum of the total squared Euclidean distances, the adaptive counterpart adaptively determines the cluster positions by using different update equations. In this experiment, the following update rule was used from [Darken and Moody, 1990]:

$$\Delta c_i = (primitive_{i+1} - c_i)/\sqrt{(i+1)}$$

where c_i is the position of the K-centre closest to the incoming primitive,

$primitive_{i+1}$. Δc_i is the increment of position that c_i must undertake, $primitive_{i+1}$ is the value of the new primitive, and i is the total number of inputs experienced previous to the new primitive. As can be seen, the operation of the adaptive version of K-means is simple and computationally inexpensive, moreover it only requires storage of the position of the k-centres and the number of data-points observed by each k-centre. Summarising, adaptive k-means clusters new primitives by (i) finding the closest cluster centre to the input primitive and (ii) using an update rule to modify the position of this cluster centre.

ITS-K-means is a novel algorithm that further extends adaptive K-means by building tree-structured clusters. The ITS-K-means algorithm is illustrated in Algorithm 2.

ITS-K-means is called every time a new primitive has to be clustered; x is the p-dimensional data-point to cluster. The tree-structure T of clusters is also given. Initially, this structure contains a unique cluster or concept, thus all primitives will be part of it. Next, the algorithm iterates until a leaf concept is found. The *closestCentre* function finds the cluster centre c, closest to primitive x that has the parent cluster *parent*. When $parent = null$ the root concept is selected. The *update* function applies the update rule previously described to modify the position of the cluster centre. At the end of the process, the tree-structure can be specialised, by adding children nodes or concepts to the representation. The following section describes the specialisation heuristic.

Algorithm 2: ITS-K-means

Input: x, p-dimensional primitive to cluster;

T, tree-structure of clusters;

Output: T, tree-structure of clusters;

```
i = 0;
parent = null;
repeat
    c = closestCentre(x, T, i, parent);
    T = update(c, x);
    parent = c;
until c = leaf;
T = specialise( T );
```

Specialisation heuristic

In order to specialise its representation, the robot must be capable of answering at least the following two questions: (i) when should the representation be specialised? (ii) which action or state concepts should be specialised?

It is assumed that the representation should be specialised when the robot using the current representation fails to achieve the desired behaviour, thus a more specialised representation (more concepts) is needed. The difficulty is to establish what causes the behaviour to perform poorly. For example a robot could be behaving poorly because: (i) the behaviour function is not yet known, (ii) the task may be accomplishable only within a certain probability of success, or (iii) the number of state or action concepts could be insufficient for solving the task. Only the third reason is related to a deficiency in the representation and only in this case should the concept representation be specialised. Once the decision to specialise the representation has been made, the problem facing the agent is to decide which state or action concept to

specialise.

In the following, an *ad-hoc* method for the specialisation of the robot's state and action representation is described. The main idea behind the method is to maintain the representation unchanged, to allow some time for the behaviour function to be learned and then specialise the concepts that are used most often.

A leaf concept c is specialised if the following criteria are satisfied:

$$\left\{ \begin{array}{l} \text{if } \; n_c \geq T_n \\ \text{if } \; \sigma_c \geq T_\sigma \end{array} \right\} \; \rightarrow \; \text{split}$$

where n_c is the number of primitives that concept c has been updated with, σ_c is the standard deviation of the observed primitives, and finally T_n, T_σ are threshold parameters.

The threshold T_n acts as an importance measure, i.e. concepts that are updated more often will also be specialised more often. Because concepts that are deep in the tree are less visited, T_n ensures that the tree does not expand excessively; it also ensures that a minimum number of primitives are observed before specialisation, ensuring that the concept is not just noise. T_σ allows one only to specialise concepts that expand a minimum 'area', thus stopping the tree from growing when a maximum degree of specialisation is achieved. These thresholds may be different for action and state spaces, and must be chosen experimentally.

5.7 Summary

This chapter has demonstrated the learning architecture presented in Chapter 4 on a simple navigational task. The robot used in these experiments had large state and action spaces. To reduce the memory necessary to represent the atomic states and actions, and to make efficient usage of training data, a method based on acquiring generalisation concepts was tested.

Generalisation concepts were acquired using a distance-based clustering technique, namely K-means. This technique creates k, disjoint clusters based on the data's distances when mapped onto a multidimensional coordinate space. The problem of selecting an appropriate scale on which to base the dissimilarity among primitives, was exemplified. To address this problem, a method based on the sensory-motor coordination effect was presented. Although the method proved of some benefit in finding the dissimilarity scales, it required the robot to have a behaviour to drive the SMC effect. Moreover, as the K-means algorithm requires the complete set of data before clustering, a data-gathering stage was necessary.

In order to make the concept generation method incremental, a novel algorithm based on K-means was presented. ITS-K-means algorithm clusters primitives as they are observed by the robot, thus there is no need to gather data in an off-line manner. Moreover, the new algorithm introduced represented 'generalisation concepts' in a tree-structure, which could be adapted to increase the number of state and action concepts. Preliminary results, reported in [Iravani, 2004] showed that the incremental and adaptive method was capable of generating a set of concepts, and that these could be used to learn the simple navigation behaviour described in this chapter.

In conclusion, this chapter has demonstrated the feasibility of acquiring generalisation concepts from the sensor and motor data of a robot using different clustering-based techniques. It also showed that state and action concepts can be used for behaviour learning and robotic control in a simple navigation task.

Chapter 6

Experimental Results: Relational Concepts for Strategic Behaviour Learning

The learning architecture presented in Chapter 4 bases behaviour learning on concepts which represent, in a compact and structured manner, the input spaces i.e. state and action spaces. Chapter 5 demonstrated how generalisation concepts can be acquired from a robot's sensor and motor data. This chapter demonstrates how relational concepts can be also be acquired from a robot's sensor and motor data. The results of this chapter were first reported in [Iravani, 2006]

Relational concepts are defined by sets of primitives that share relations among them. For example, in Chapter 4 an *arch* was introduced as a relational concept requiring a set of blocks and a relation among these blocks to be defined. This chapter will show how the Q-analysis method presented in

Section 3.4 can be used to generate relational concepts.

6.1 Experimental test-bed

This section describes the test-bed used for generating relational concepts and learning team-strategic behaviours. It is based on the RoboCup simulator briefly described in the following.

6.1.1 RoboCup simulation league

The RoboCup simulation league is a fully distributed multiagent domain in which autonomous agents participate in a cooperative and competitive football game [Noda *et al.*, 1998].

The environment is a simulated football game with eleven players in a team. At any point in time, each agent can choose among three actions, these are: dash, turn and kick. The dash and turn actions are mainly used for robot navigation, while kick is used to control, pass and shoot the ball. By combining these actions and their sensory perception, agents must be capable of cooperating with team-mates in order to score more goals than the opposition team.

The simulator has been conceived to be as similar as possible to 'real world' robot football. For this reason, sensors and actuators have random noise, observations are limited in the environment, and agents even get tired.

These characteristics make the RoboCup simulation league a complex and realistic multiagent test-bed. A comprehensive description of the simulator and the simulation league can be found in [Stone, 1998, Chen *et al.*, 2002].

6.1.2 Ball-passing behaviour

At any point in time, the player in possession of the ball must decide whether to dribble, pass or shoot the ball. Thus, ball-passing is one of the three most important behaviours in robot soccer. If a player decides to pass the ball, it must also choose which team-mate player to pass it to.

In this experiment, the *ball-passing* behaviour is defined as the behaviour that selects a team-mate player to pass the ball to. The passing behaviour is considered successful if it selects a player and results in a successful pass, otherwise it is considered a failure.

The main aim of the experiment is to generate a set of relational concepts from which the ball-passing behaviour can be learned. In other words, finding a set of relational concepts that indicate whether a team-mate player is well situated for receiving a pass or not.

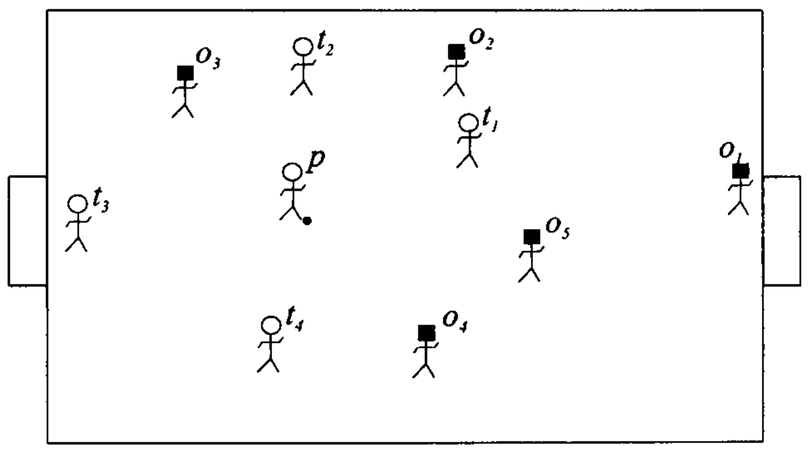
The concepts related to the ball-passing behaviour will be learned from historical data, by observing successful passes in games played in the past. The next section describes how the data is gathered.

6.1.3 Pass data gathering

The simulator server produces *log-files* of all games played, i.e. it records the positions of all players and the ball at all times during a game. In this experiment, the log-files of the RoboCup 2003 competitions were used, in particular that of the final game between the teams, *UvA Trilearn* and *Tsinghuaelous*.

In order to learn the concepts related to the passing behaviour, the log-file is pre-processed to extract passing scenes. A *passing scene* is a static view

of the field at the time the robot in possession of the ball is ready to start a pass. Each passing scene contains information on the positions of all the players, and the receiver of the pass. Figure 6.1(a) illustrates a passing scene, where p is the player which must execute the pass, $t_1, t_2, \ldots t_4$, are its team-mate players, and $o_1, o_2 \ldots o_5$, are players of the opposition team. Figure 6.1(b) illustrates the format in which the scene information in given, where the position of each player is given by their (x, y) coordinates, and *receiver* indicates the team-mate player which received the pass in that scene. For clarity, Figure 6-1 illustrates five players per team, rather than the eleven players that are used in the RoboCup Simulator.



(a) An example of a passing scene

player	position	receiver
p_1	(p_x, p_y)	0
t_1	(t_{1x}, t_{1y})	0
t_2	(t_{2x}, t_{2y})	0
t_3	(t_{3x}, t_{3y})	1
t_4	(t_{4x}, t_{4y})	0
o_1	(o_{1x}, o_{1y})	0
$\vdots$	$\vdots$	$\vdots$
o_5	(o_{5x}, o_{5y})	0

(b) Data of a passing scene

Figure 6-1: A passing scene and the data related to it

6.2 State in the ball-passing behaviour

The navigation behaviour in the previous chapter used a simple state description, i.e. the combination of the distance and angle values with respect to the target, that two-dimensional state representation proved sufficient for

controlling a robot in the navigation task. The ball-passing behaviour is more complex, i.e., it depends on many more variables and their interrelations. For example, to select a team-mate player to pass the ball to, the passer must base the choice on the state of all of its team-mates. The state of a team-mate can be composed of variables such as: the position of the team-mate, the distances from its opponents, its position in the field, its area of possession, etc. In principle, there is no obvious best-set of variables to describe this state. In this context, the experiment followed the method of using a wide range of variables (some of them possibly irrelevant) to describe the state, and allowing the concept generation method to generate concepts containing only relevant variables, i.e. by feature selection.

The remainder of this section defines the variables used to describe the team-mate players' states, and how these states can be represented using an incidence matrix and a simplex representation.

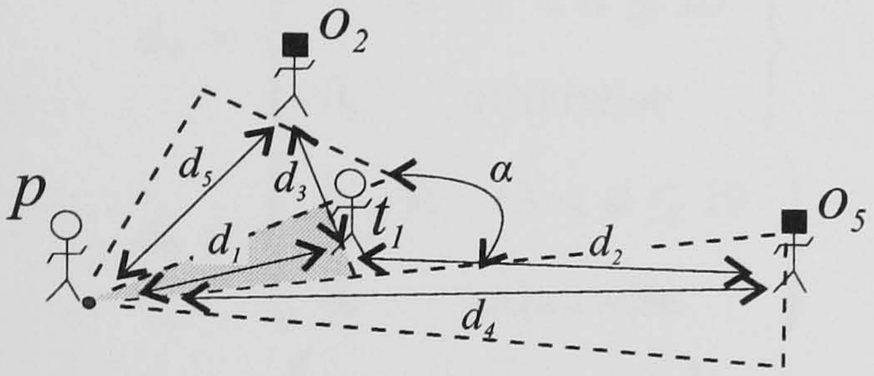
6.2.1 State variables

The number of variables that could be used to describe the state in a robot soccer game is very large. This section explains the variables, arbitrarily selected, needed to describe the state in the ball-passing experiment.

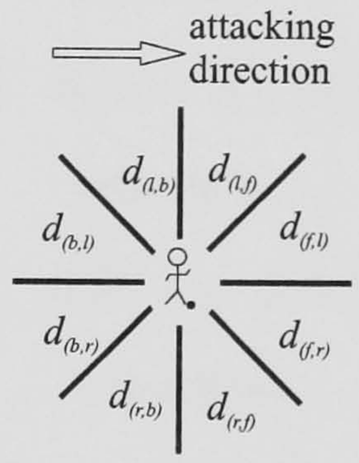
The state of each of the team-mate players is described by the following fifty binary variables:

- five distances: $d_1, d_2, \ldots, d_5$
- one angle: α
- four neighbour relations: $RN, R\bar{N}, LN, L\bar{N}$

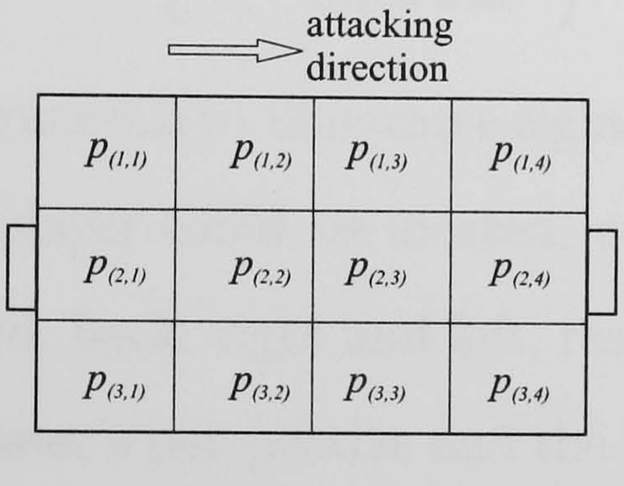
- two relations to the closest opponent: $OPP, O\bar{P}P$
- eight pass directions: $d_{(f,l)}, \ldots, d_{(b,l)}$
- twelve field positions: $p_{(1,2)}, \ldots, p_{(3,4)}$



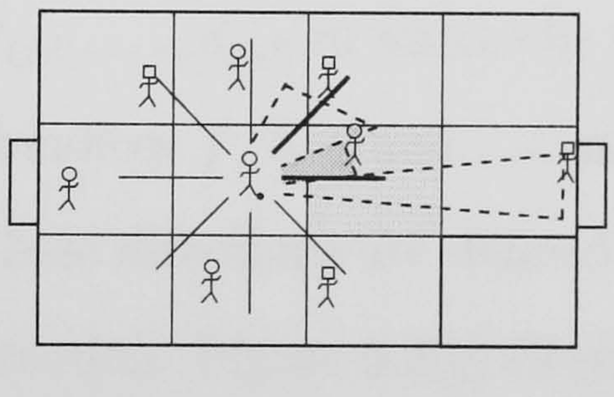
(a) A team-mate's area



(b) A team-mate's direction



(c) A team-mate's position



(d) Team-mate state representation

Figure 6-2: State representation

Some of these variables are illustrated in Figure 6-2. For instance, Figure 6.2(a) illustrates a set of variables related to the players's controlled area. This area is defined by the distances $d_1, \ldots, d_5$ and the angle α between the four players seen in the figure. These players are: the passer p, the team-mate being described t_1, and its two neighbouring players o_2 and o_5, which in this case belong to the opposite team. The distances and the angle are continuous variables and have been segmented into four binary intervals: 'very-small',

'small', 'big', and 'very-big', denoted by vs, s, b, vb. The following illustrates the arbitrary segmentation thresholds used:

$$d_{vs} = \left\{ \begin{array}{ll} 1, & \text{if} \quad 0 < d \leq 7 \\ 0, & \text{otherwise} \end{array} \right\} \qquad \alpha_{vs} = \left\{ \begin{array}{ll} 1, & \text{if} \quad 0 < \alpha \leq 0.15 \\ 0, & \text{otherwise} \end{array} \right\}$$

$$d_{s} = \left\{ \begin{array}{ll} 1, & \text{if} \quad 7 < d \leq 13 \\ 0, & \text{otherwise} \end{array} \right\} \qquad \alpha_{s} = \left\{ \begin{array}{ll} 1, & \text{if} \quad 0.15 < \alpha \leq 0.35 \\ 0, & \text{otherwise} \end{array} \right\}$$

$$d_{b} = \left\{ \begin{array}{ll} 1, & \text{if} \quad 13 < d \leq 19 \\ 0, & \text{otherwise} \end{array} \right\} \qquad \alpha_{b} = \left\{ \begin{array}{ll} 1, & \text{if} \quad 0.35 < \alpha \leq 0.70 \\ 0, & \text{otherwise} \end{array} \right\}$$

$$d_{vb} = \left\{ \begin{array}{ll} 1, & \text{if} \quad d > 19 \\ 0, & \text{otherwise} \end{array} \right\} \qquad \alpha_{vb} = \left\{ \begin{array}{ll} 1, & \text{if} \quad \alpha > 0.70 \\ 0, & \text{otherwise} \end{array} \right\}$$

Figure 6.2(b) illustrates eight directions $d_{(f,l)}, \ldots, d_{(b,l)}$ in which the team-mate player could be located, where the subindices f, b, r and l stand for: forward, back, right and left, respectively. These directions are defined from the passer's perspective and the attacking direction. Figure 6.2(c) illustrates the playing field divided in twelve positions ($p_{(1,2)}$, ..., $p_{(3,4)}$), in which a team-mate player could be located. The subindices relate to the position in the field as seen in the figure. These positions are also relative to the attacking direction. Figure 6.2(d) illustrates how the variables are measured with respect to a team-mate player (in grey).

The variables not shown in the figure, i.e., RN, $R\bar{N}$, LN, $L\bar{N}$, OPP and $O\bar{P}P$, have the following meaning. The value of RN (right neighbour) and LN (left neighbour), is 1 if the neighboring players of the player being described, are also team-mates. For example, in Figure 6.2(a) RN and LN would be 0, as both neighbouring players o_1 and o_5, belong to the opposing team. The value of opponent closer (OPP) is 1 if a neighbouring players is an

opponent, and if it is closer to the ball. In Figure 6.2(a), OPP would be 0, as both opponent neighbours are further from the passer than the team-mate.

The variables, $R\bar{N}$, $L\bar{N}$ and $O\bar{P}P$, represent the negation of RN, LN and OPP. These negated values are known in Q-analysis as *anti-vertices*. Anti-vertices represent the case that two elements are *not* related. That is, if a sensor is not active with respect to a state, then the anti-vertex representing that relation will be a 1 in the incidence matrix. Representing non-existing relations is interesting in some situations. For example, for the ball-passing behaviour, the fact that an opponent is not closer to the ball may be of interest.

Table 6.1: Summary of the state variables

variable	description	binary values
d_1	distance passer to team-mate	4
d_2	distance team-mate to right neighbour	4
d_3	distance team-mate to left neighbour	4
d_4	distance passer to right neighbour	4
d_5	distance passer to left neighbour	4
α	angle receivers area	4
RN	right neighbour is team-mate	1
$R\bar{N}$	right neighbour is not team-mate	1
LN	left neighbour is team-mate	1
$L\bar{N}$	left neighbour is not team-mate	1
OPP	opponent is closer to ball	1
$O\bar{P}P$	opponent in not closer to ball	1
$d_{(f,l)} \cdots d_{(b,l)}$	pass direction	1×8
$p_{(1,2)} \cdots p_{(3,4)}$	pass position	1×12

Table 6.1 presents a summary of these variables, and includes a short description of each variable with its number of binary values.

Some of the variables are mutually exclusive. For example, if d_1 has value vs it can not have any of the remaining three values s, b and vb. Thus, the state of each team-mate is described by eleven active binary variables, thus each team-mate player can be represented by a 10-simplex, as shown in the next section.

The previous definition of the team-mates' state assumes full observability of the game, i.e. the passer can perceive all the information above. This assumption does not hold when the players are competing in the game, as they only have a partial view of the field depending on the direction they are facing. The assumption of full observability is made on the basis that most teams have incorporated players architectures which acquire models of the whole field. For example, see the player architecture developed in [Stone, 1998].

6.2.2 Incidence matrix representation of the state

As introduced in Section 3.4.2, Q-analysis represents multidimensional data using an incidence matrix representation. Each of the states of a team-mate player can be represented as a row in an incidence matrix.

Table 6.2: Incidence matrix representation of a team-mate player

team-mate	d_1				...	d_5				α				RN	$\bar{RN}$	...
t_1	0	0	1	0	...	1	0	0	0	0	1	0	0	0	1	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	...

LN	$L\bar{N}$	OPP	$O\bar{P}P$	$d_{(f,l)}$	...	$d_{(b,l)}$	$p_{(1,2)}$	...	$p_{(3,4)}$
0	1	1	0	1	...	0	0	...	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

An example of such an incidence matrix is given in Table 6.2. As described above, only eleven of the fifty columns will contain a '1'. In other words, only 11 of the variables are related to any team-mate player.

6.2.3 Simplex representation of the state

As described in Section 3.4.3, each row of an incidence matrix can be represented by a simplex. Since the team-mate state has eleven active variables, each row of the matrix will determine a 10-simplex.

6.3 Relational concepts in the ball-passing behaviour

This section applies the method exemplified in Sections 3.5 and 3.6 to the data from the RobCup Simulator. Section 6.1.3 described the passes data-set. Section 6.2 defined the state description for each one of the team-mate players which are summarised in Table 6.1. There, it was shown that a 10-simplex represented the state of each team-mate player. In each passing scene, there is one team-mate which receives the pass (receiver) and nine which don't (non-receiver) (see Figure 6.1(b)).

In this experiment, the first question asked is whether any structural difference exists between the simplices representing 'receivers' team-mates, and those representing 'non-receivers' team-mates. To investigate this question, the star-hub analysis is used to identify which are the hubs that occur most in the case of receivers. The resulting hubs will then be contrasted with the simplices representing non-receivers.

If there exist any structural differences between receivers and non-receivers these can be exploited to define concepts related to good and bad passing configurations. This is investigated in the next section.

6.3.1 Structural differences between receivers and non-receivers

The star-hub analysis was first applied to the simplices representing the configuration of receivers. The RoboCup 2003 Final game, contains 260 passing scenes. Of those, 118 correspond to the winning team, i.e. UvA Trilearn. The analysis was carried out on these last passing scenes, thus there were 118 simplices representing receiver team-mates, and $9 \times 118 = 1062$ simplices representing non-receiver team-mates.

The star-hub analysis applied to the receiver simplices produced over 3000 hubs. From these hubs, some of those with most simplices, i.e. the ones that occurred most often, were selected, Table 6.3 illustrates them.

Table 6.3: A selection of hubs for the pass data

hub	receiver	non-receiver
$\langle O\bar{P}P \rangle$	70%	50%
$\langle \bar{RN}, O\bar{P}P \rangle$	43%	24%
$\langle d_1(s), \bar{LN} \rangle$	36%	15%
$\langle \bar{RN}, \bar{LN}, O\bar{P}P \rangle$	24%	10%
$\langle d_1(s), \bar{RN}, \bar{LN} \rangle$	20%	9%
$\langle d_1(s), R\bar{N}, L\bar{N} \rangle$	20%	9%
$\langle d_1(s), d_5(vb), \alpha(vs), O\bar{P}P \rangle$	15%	4%
$\langle d_1(vs), \bar{RN}, LN, O\bar{P}P \rangle$	8%	2%
$\langle d_2(s), d_3(s) \rangle$	13%	10%
$\langle d_3(s), d_4(b) \rangle$	9%	7%

The selected hubs are then used on the non-receiver team-mates data to measure their frequency of occurrence. As can be seen in Table 6.3, there are some hubs that occur more in relation to receiver players than to non-receiver ones. For example, $\langle O\bar{P}P\rangle$ occurs with a frequency of 70% in the receivers data and 50% in the non-receivers data. This indicates that the hub is more related to good passing situations. This result seems plausible, as not having an opponent closer ($O\bar{P}P = 1$) allows passing with a lower risk of ball interception.

This shows that some structural differences exist between the configuration of receiver and non-receiver team-mate players. In a similar experiment conducted with fewer descriptive variables similar conclusions were reached [Iravani *et al.*, 2005].

The last two hubs in the table show a similar probability of occurrence in both classes (low specificity), thus in principle, if these variables do not appear in other relevant hubs (higher specificity), they could be considered as irrelevant in relation to the concepts of 'receiver' and 'non-receiver' players.

6.3.2 Study of the effect of neighbouring players

As described in Section 6.2.1 the state of a team-mate player includes the relations to its neighbouring players, namely: $RN, R\bar{N}, LN, L\bar{N}, OPP, O\bar{P}P$. Table 6.4 illustrates the percentage of 'receiver' and 'non-receiver' simplices that share the hubs related to the previous variables.

The hubs of dimension $q = 0$ represent the probability of observing each of these in the data. That is, 33.1% of the receiver simplices have the hub $\langle RN\rangle$, against 42.2% of the non-receiver. By looking at the first four hubs,

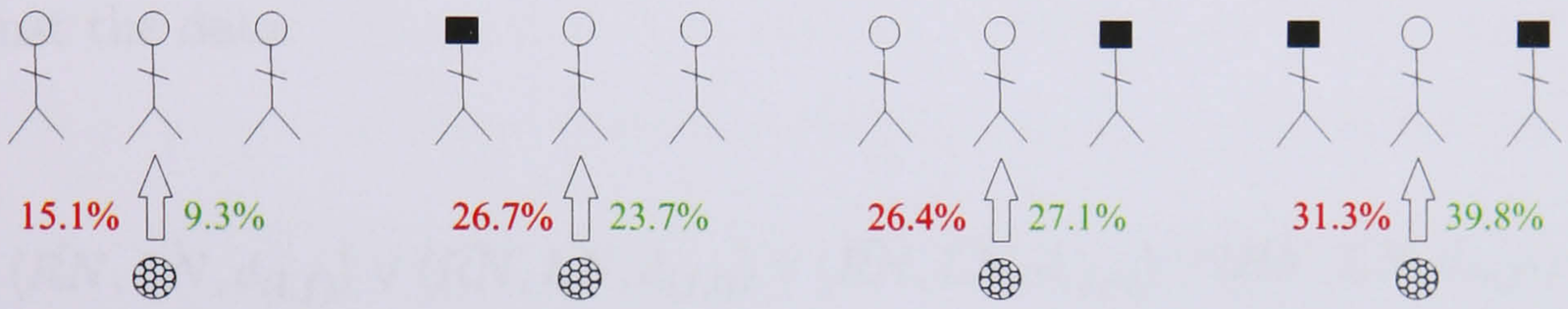
we find that $\langle R\bar{N}\rangle$ and $\langle L\bar{N}\rangle$ have higher probability of occurrence in both receiver and non-receiver classes. This means that in the game's passing scenes it is more common for team-mates to have opponents as neighbours. This is plausible, as in most cases the defending team will be covering the players of the attacking team.

Table 6.4: Neighbour player relation

hub	receiver	non-receiver
$\langle RN\rangle$	33.1%	42.2%
$\langle R\bar{N}\rangle$	66.9%	57.8%
$\langle LN\rangle$	36.4%	41.9%
$\langle L\bar{N}\rangle$	63.5%	58.1%
$\langle RN, LN\rangle$	9.3%	15.1%
$\langle RN, L\bar{N}\rangle$	23.7%	26.7%
$\langle R\bar{N}, LN\rangle$	27.1%	26.4%
$\langle R\bar{N}, L\bar{N}\rangle$	39.8%	31.3%
$\langle R\bar{N}, L\bar{N}, O\bar{P}P\rangle$	24.1%	20.9%
$\langle R\bar{N}, L\bar{N}, OPP\rangle$	15.7%	10.4%

The relations represented by the hubs of dimensions $q = 1$ are illustrated in Figure 6-3; the percentage of receiver and non-receiver simplices are also indicated, in green and red, respectively. Figure 6.3(a) illustrates a situation in which the receiver team-mate was surrounded by players of its own team. Figures 6.3(b) and Figure 6.3(c) illustrate a situation in which the receiver team-mate is surrounded by a team-mate and an opponent. Figure 6.3(d) illustrates a situation in which the receiver team-mate is surrounded by opponents. The hub $\langle RN, LN\rangle$, (Figure 6.3(a)) seems to describe the best state for passing as both neighbours are of the same team, i.e. there is low probability of interception by the opponents. But their probability of

occurrence tells us the contrary, that is, the passer chooses more often not to pass to a team-mate which is in this configuration. Similar information is encoded in hub $\langle R\bar{N}, L\bar{N}\rangle$, which indicates that both neighbours of the team-mate player are of the opposing team (Figure 6.3(d)). In this case, it would appear that it is better not to pass as there are two opposition players near (risky pass situation). But the probabilities associates with this hub tells us that it is more probable to pass to a team-mate in this situation. The probabilities associated with these two hubs seem counter-intuitive.



(a) $\langle RN, LN\rangle$ (b) $\langle RN, L\bar{N}\rangle$ (c) $\langle R\bar{N}, LN\rangle$ (d) $\langle R\bar{N}, L\bar{N}\rangle$

Figure 6-3: Neighbouring relations

Why does a player prefer to pass to a team-mate that has many opponents near? Why does the player prefer not to pass in easy-passing situations?

A possible answer is as follows: easy passing situations, such as the one in Figure 6.3(a), happen only in a direction for which the pass is not desired, for instance, in passing backwards. To test this hypothesis, the following hubs have been tested against the data:

$$\langle RN, LN, d_{(l,b)}\rangle \vee \langle RN, LN, d_{(b,l)}\rangle \vee \langle RN, LN, d_{(b,r)}\rangle \vee \langle RN, LN, d_{(r,b)}\rangle$$

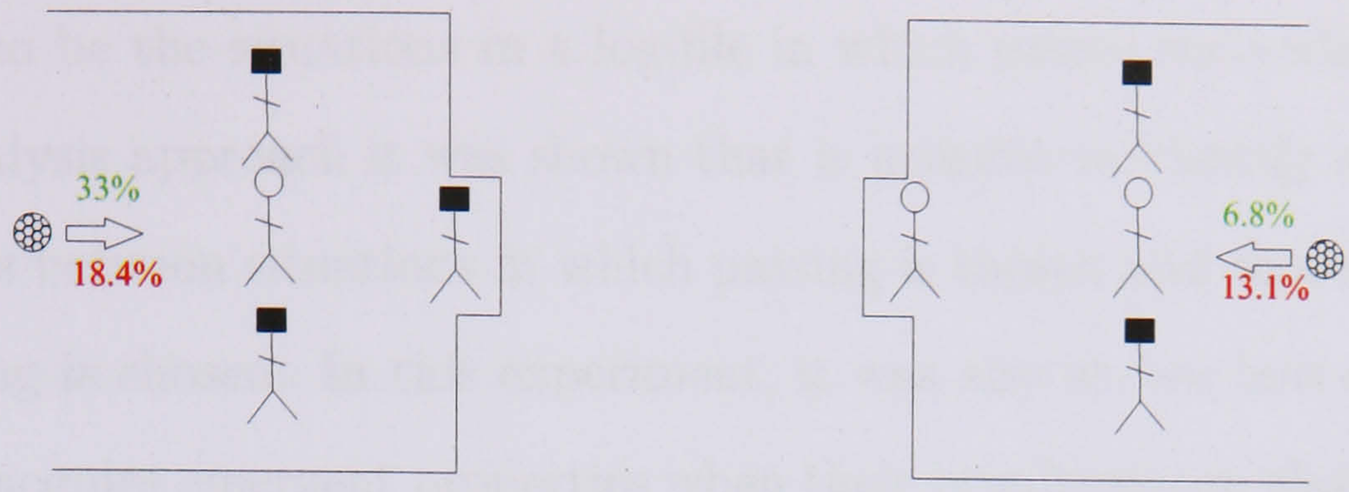
Any simplex containing these hubs would represent a simple pass situation (RN, LN), towards the back direction (towards its own side of the field). See

Figure 6.2(b) for a description of these directions. Applying these hubs on the data results in the following: 7.6% out of 9.3% (81.7%) of the receiver simplices and 13.4% out of 15.1% (88.2%) of the non-receiver simplices are towards the back. This corroborates the hypothesis that easy-passing situations are not chosen because they mostly occur (81.7% and 88.2%) towards the team's own side of the field.

To test the counter hypothesis, that is, that risky passes towards the opponent's goal are chosen for passing, the following hubs have been tested against the data.

$$\langle \bar{RN}, \bar{LN}, d_{(l,f)} \rangle \vee \langle \bar{RN}, \bar{LN}, d_{(f,l)} \rangle \vee \langle \bar{RN}, \bar{LN}, d_{(f,r)} \rangle \vee \langle \bar{RN}, \bar{LN}, d_{(r,f)} \rangle$$

Simplices satisfying these hubs indicate that both the team-mate's neighbours are opponents, and that the direction of the pass is towards the opponent's goal. Applying these hubs results in: 33% out of 39.8% (82.9%) of the receiver simplices having two opponent as neighbours, and the pass directed towards the opponent's goal. This indicates that although the pass is risky, it is undertaken because it moves the ball in the attacking direction. 18.4% out of 31.5% (58.4%) of the non-receiver simplices have this configuration. This indicates that fewer non-receiver simplices have this configuration, thus the configuration of two neighbouring opponents and a direction towards the opponent's goal corresponds more to a better passing situation. Figure 6-4 illustrates this configurations, together with receiver and non-receiver probabilities.



(a) 'Risky' pass going forward (b) 'Risky' pass going backward

Figure 6-4: 'Risky' pass

These results indicate that what seemed counter-intuitive situations, i.e., passing to players in risky situations and not passing to players in easy situations, could be explained when taking into account the direction of the pass. This illustrates how adding vertices into hubs results in concepts having emergent properties. That is, an easy passing situation becomes a non-passing situation when the pass is directed towards the team's own goal, whereas a risky pass situation becomes a pass when this is directed towards the opponent's goal.

6.4 Summary

This chapter has shown how to use Q-analysis to generate relational concepts. Data extracted from the log-files of the RoboCup Simulator were used to demonstrate the classification method introduced in Sections 3.5 and 3.6 on robotic data. The aim of the experiment was to acquire relational concepts in the context of a ball-passing behaviour, in other words, generating concepts that represented when a team-mate player was in a 'good situation' to receive a pass and when it was not. Good situations for receiving a pass were

assumed to be the situations in a log-file in which passes succeeded. Using the Q-analysis approach it was shown that is possible to identify structural differences between situations in which passing is chosen and those in which not-passing is chosen. In this experiment, it was also shown how relational concepts acquire emergent properties when their primitives are classified, for example, how an apparently easy passing situation can be not selected if the direction of the pass is towards the team's own side of the field.

A complete analysis, including the heuristic method for finding classifier hubs, was not applicable in ball-passing. The reasons are the following:

- Computational intensive process: The star-hub analysis presented in this chapter is computationally intensive, that is, finding the hubs in the data requires all hubs to intersect with each other. This results in a relatively small data-set, containing 118 receiver primitives and 1062 non-receiver ones, having a huge number of hubs, approximately 58000.

- Hubs sparsity: In such a large hub space, most hubs are only shared by two simplices. Thus, they do not provide the reliable statistics of specificity and broadness needed to define classifier hubs.

- Inconsistency in the data: Some of the team-mates labelled as non-receivers share similar variables to those labelled as receivers. This is because it is possible for many team-mates to be in a 'good passing' situation, but only one of them can be the pass receiver.

In conclusion, we can say that the method presented has promising characteristics for classification and the generation of relational concepts. The following summarises these characteristics:

- The structural similarity exploited by this method does not suffer from the so-called *chalk and cheese* problem. As the similarity or dissimilarity of primitives is not reduced to a distance measure, it is not necessary to 'scale' the relevance of each dimensions.

- The heuristic method presented for defining relational concepts or classifier hubs shows interesting characteristics in relation to filtering irrelevant variables. That is, irrelevant variables in relation to a class or concept can be eliminated, reducing the total number of dimensions, and also addressing partially the problem of the curse of dimensionality.

- As seen in Section 6.3.2, relational concepts, i.e. classifier hubs can be easily interpreted by the designer. That is, any of the hubs studied could be easily mapped back onto the original data, and their meaning also be easily understood.

- The heuristic presented for discovering classifier hubs is computationally expensive and only applicable to small data-sets. In order to apply the ideas from the Q-analysis methodology into the classification of large data-sets, more efficient heuristics would be needed.

Chapter 7

Conclusions

Motivated by the challenges of designing 'truly' autonomous agents, this thesis addresses two key aspects of autonomy, those of learning and adaptation. This chapter states the conclusions that can be drawn from this thesis, and also proposes future research directions that emerge from it.

7.1 Answers to the research question

The various arguments made through this thesis (summarised in Chapter 1) are now revisited, based on the research questions addressed.

- Question **Q1: Is it possible to use robotic sensor and motor data to learn abstract entities called *concepts*?**

  The thesis has given an affirmative answer. Chapter 3 introduced concepts as classes formed by multidimensional primitives. The chapter also reviewed some of the existing techniques for multidimensional data classification, discussed their main limitations and proposed a novel

classification technique based on the methodology of Q-analysis. Two examples were conducted to illustrate the functionality of the classification technique, one based on the synthetic CorrAL data-set and the other on Fisher's Iris data.

Later, in Chapter 5 and Chapter 6, experimental evidence was provided to show how concepts can be learned by classifying observed sensor and motor data into state and action concepts.

In Chapter 5, data collected from a robot interacting with its environment were used as primitives to generate concepts. More precisely, the states and actions observed by the robot were classified using distance based clustering techniques; this classification resulted in what we called *generalisation concepts.*

In Chapter 6, data from the RoboCup Simulation League was used as primitives to generate *relational concepts.* The classification method based on Q-analysis was used to classify primitives according to a similarity metric based on relational structures.

- Question **Q2: Is it possible to use such *concepts* as the representation for learning robotic behaviours? Do such *concepts* provide any benefits for behaviour learning? In particular, how do they address the generalisation problems faced in machine learning?**

  The thesis gives an affirmative answer. In Chapter 5, concepts representing the state-action spaces of a mobile robot were used as the representation for learning a navigation behaviour.

Behaviour learning was based on finding a function that mapped state concepts into action concepts. In that particular example, the state-action spaces were two-dimensional spaces, and the number of possible states and actions combination was 432×10^6. Using this large state-action space a more compact space was defined based on clustering primitives into state and action concepts. In the first experiment, 10 state concepts and 10 action concepts were generated. Their combination resulted in 100 state-actions. The behaviour function used to learn the navigation behaviour needed only 100 memory positions to be represented, in contrast with the 432×10^6 that would have been necessary if the original state-action spaces were used. This showed that using *generalisation concepts* to represent the state and the action spaces of a robot reduced the memory requirements to represent behaviour functions. Thus, these types of concepts can be used to represent large state-actions spaces in a generalised manner.

Chapter 6 presented an experiment where *relational concepts* were generated for learning a strategic ball-passing behaviour. The main idea in the experiment was to observe successful passes and to generate concepts representing the situations in which passing would be successful. Thus, the robot learned some situations in which passing was successful.

- Question **Q3: Is it possible to control autonomous mobile robots using this notion of *concept*?**

  The thesis gives an affirmative answer. Chapter 5 demonstrated that using a navigation behaviour represented by generalisation concepts it

is possible to control a robot in a simple navigation task. The control was based on observing the state of the environment, classifying this state into its corresponding state concepts, then using the behaviour function to select an appropriate action concept and use this as the control action.

Chapter 6 showed how a soccer-robot could learn relational concepts in relation to a ball-passing behaviour. Assuming that a robot had a set of concepts describing good passing situations, then when one of these would be observed in a game, the robot would know that a good passing option was available. Higher-level decision making could then decide whether to pass or execute any other action. This type of control was not demonstrated in this thesis, and is a good candidate for future investigation.

- Question **Q4: Is it possible to integrate the notion of *concept* within a multilevel architecture which exploits the definition of concepts for learning and control?**

  The thesis gives an affirmative answer. Chapter 4 presents an architecture that generates concepts by hierarchically classifying primitives. These concepts are then used as the representation for learning control behaviours. The proposed architecture divides the behaviour learning task into two inter-related sub-tasks.

  The first sub-task is to learn a representation of the state and action spaces based on learning state and action concepts. This sub-task was implemented by an architecture's component known as concept generation. This component applies different classification techniques to

generate a multilevel hierarchy of concepts. In Chapter 5 concepts were generated using different clustering techniques, whereas in Chapter 6 a novel method based on classification using Q-analysis was used.

The second sub-task is to learn a behaviour function represented using the previously generated concepts. This sub-task was implemented by the behaviour learning and control component. A supervised learning approach was taken to learn behaviour functions. That is, a set of examples of the desired behaviour was provided for the robot, which had to learn an appropriate behaviour function so that a similar behaviour to the one exhibited by the examples is attained.

The behaviour learning and control component is also used to control the robot. Control is done in a reactive manner, where states are classified into state concepts and the behaviour function reactively chooses the adequate action concept.

Chapter 5 demonstrated the functionality of the architecture for a simple navigation task. Chapter 6 demonstrated the functionality of the same architecture in a more complex and strategic ball-passing behaviour.

7.2 Thesis contributions

This thesis has made four major contributions in the field of multilevel learning and robotic control as follows.

- A simple multilevel architecture for robots is proposed. The architecture is based on generating concepts and then using them for learning

control behaviours. An unexpected outcome of the research is that concepts can be formed by following two different kinds of classification methods. In the first, the classification is cluster-based, and essentially set theoretic. That is, sets of primitives are classified together as concepts; these were called generalisation concepts. In the second case, the classification is based on relational structures, with primitives classified into structured concepts at a higher-level in the hierarchy; these were called relational concepts.

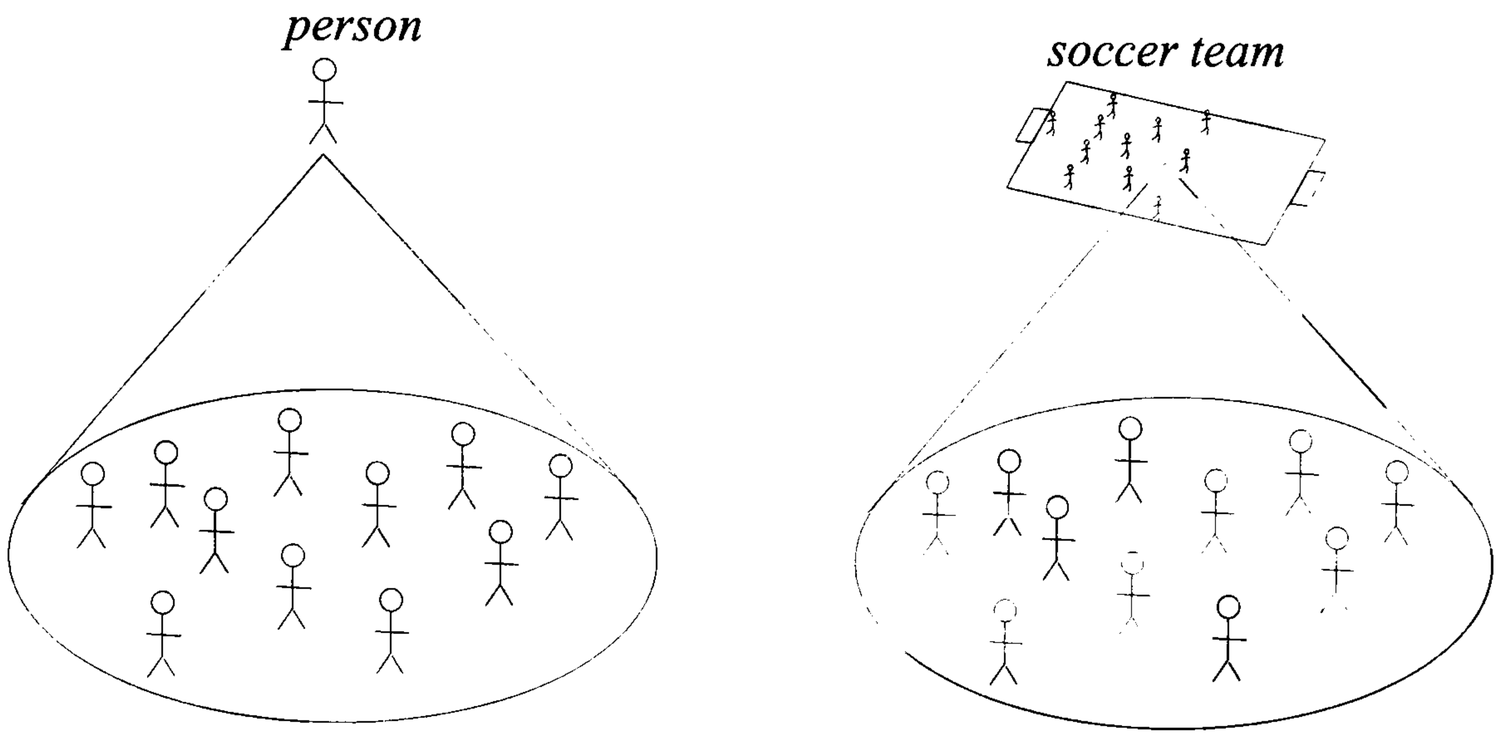
- This thesis demonstrates that it is possible to generate concepts using different classification techniques, and that the resulting concepts can be used for behaviour learning and robotic control.

- This thesis has identified that both generalisation and relational concepts are necessary for building hierarchies of concepts in the robotic domain. Generalisation concepts allow us to represent a set of primitives as a unique concept, thus the number of concepts can be considerably smaller than the number of possible primitives. As has been shown, in robotics, sensors and motors usually provide relatively large ranges of possible values. In some cases, taking groups of values as equivalent, reduces the total number of possible values, but still allows the definition of satisfactory controllers. Thus, the usage of generalisation concepts in the robotics domain allows to compactly represent sensor and motor values. Relational concepts are more interesting in the sense that they acquire emergent properties when classified using relational structures. For example, as seen in the ball-passing behaviour, an easy passing situation such as having a receiver team-mate sur-

rounded by other team-mates has the emergent property of being a good passing situation if the direction of the pass is going towards the opponent's goal, and a bad passing situation if the direction of the pass is going backwards. That is, an easy passing situation in combination with a desired passing direction has an emergent property which none of its parts have when taken in isolation. Relational concepts allow robots to discover and represent structures that capture some of the emergent properties that occur in their environments.

- This thesis reveals that concepts are at a higher descriptive level than the primitives they came from. That is, concepts are higher in the hierarchy than their primitives. However it is important to make the distinction in the way generalisation and relational concepts go up the hierarchy. That is, generalisation concepts have the same characteristics as any of their primitives, thus they do not add any new information. On the contrary, relational concepts contain emergent properties. Hence, generalisation concepts go up the hierarchy by generalising primitives, while relational concepts go up the hierarchy by creating new properties. For example, several individuals can be classified into a generalisation concept such as *person* (Figure 7.1(a)), or into a relational concept such as *soccer team* (Figure 7.1(b)). The concept person has the same characteristics of any of the individuals, while the concept soccer team has emergent characteristics, such as being capable of playing soccer. Both of these types of concept are necessary in the robotic domain, using generalisation concepts to reduce the total number of elements in the space, and relational concepts to abstract

and extract new properties from the robot, the environment and their interactions.



(a) Concept *person* (b) Concept *soccer team*

Figure 7-1: Example of generalisation and relational concepts

This thesis has also made two contributions to the field of multidimensional data classification as follows.

- It presents a novel algorithm that extends the traditional K-means clustering algorithm into an iterative and adaptive version, known as ITS-K-means. The new algorithm clusters data iteratively, that is, inputs are clustered *on the fly* when they are observed. Moreover, the new algorithm does not need to predefine the number of cluster centres or k. The number of cluster centres are adaptively acquired by specialising existing centres and organising them in a tree structure.

- It presents a novel classification technique based on the methodology of Q-analysis which is used to define a novel similarity metric. Most of the existing classification techniques assume that Euclidean distance

between data points is equivalent to the similarity between the data points. The thesis has argued that this assumption can only be made when the mathematical properties of the multidimensional spaces are well-known. That is, different dimensions can be compared only if their relations are known. The thesis presents a novel similarity metric based on the structural connectivity of the multidimensional data points, that is, by comparing the shared features that describe the data points. The experimental results indicate that the new similarity metric can be used for classification. Moreover, the new classification technique allows filtering out irrelevant features describing the data inputs.

7.3 Further work

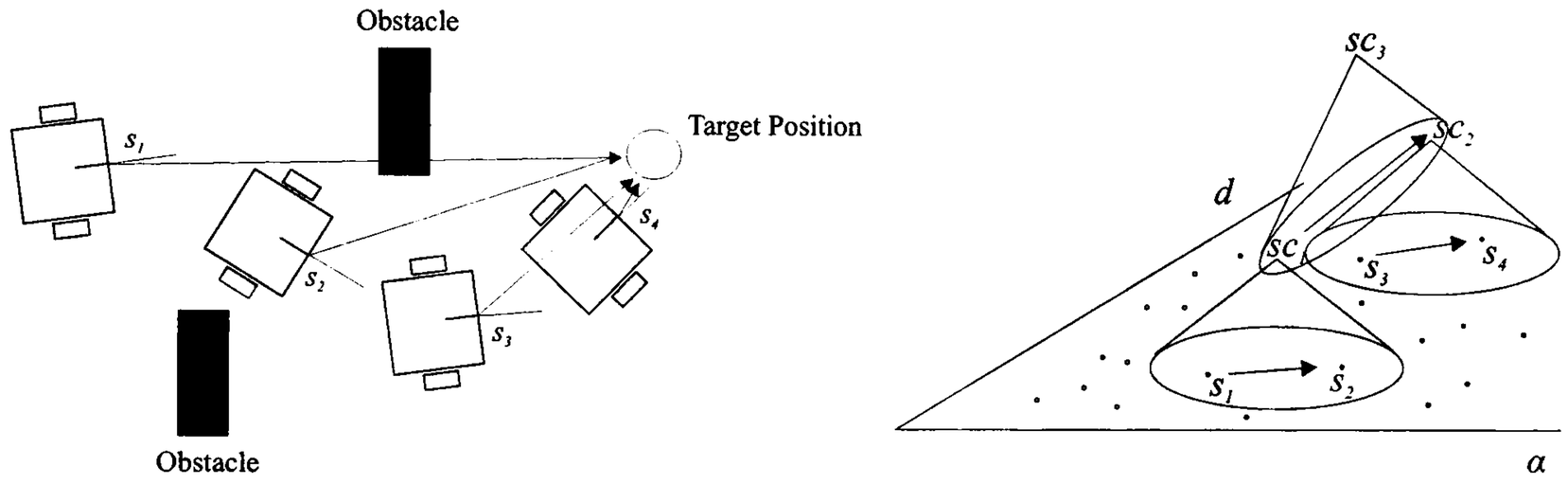
In conducting this research many interesting issues emerged which could not be addressed because of time limitations. This section describes these issues and proposes them as possible future research directions.

7.3.1 Towards goal-directed behaviour using relational concepts

Section 4.4.2 described how states can be classified into relational concepts. There, the relation used to classify primitives was the co-occurrence of sensors values, i.e. logical *AND* operations among the sensors' values. Other relations could also be exploited, such as *temporal* relations.

Temporal relations are based on the ordered sequence of state occurrences. If these relations are used to define state concepts, then these represent or-

dered histories of primitive states. For example, Figure 7.2(a) illustrates a mobile robot navigating towards a target position. Let the states s_1, s_2, s_3 and s_4 be the atomic states perceived by the robot during navigation towards the target. The robot's state history from its initial position to the target can be represented by the following state concepts: $sc_1 = \langle s_1, s_2; R_1 \rangle$ and $sc_2 = \langle s_3, s_4; R_2 \rangle$ or $sc_3 = \langle sc_1, sc_2; R_3 \rangle$, illustrated in Figure 7.2(b). Relations R_1, R_2 and R_3 represent the temporal transitions between states, i.e. $R_1 = s_1 \rightarrow s_2$, $R_2 = s_3 \rightarrow s_4$, $R_3 = sc_1 \rightarrow sc_2$, in Figure 7.2(b). These relations are represented by arrows.



(a) Robot navigation history

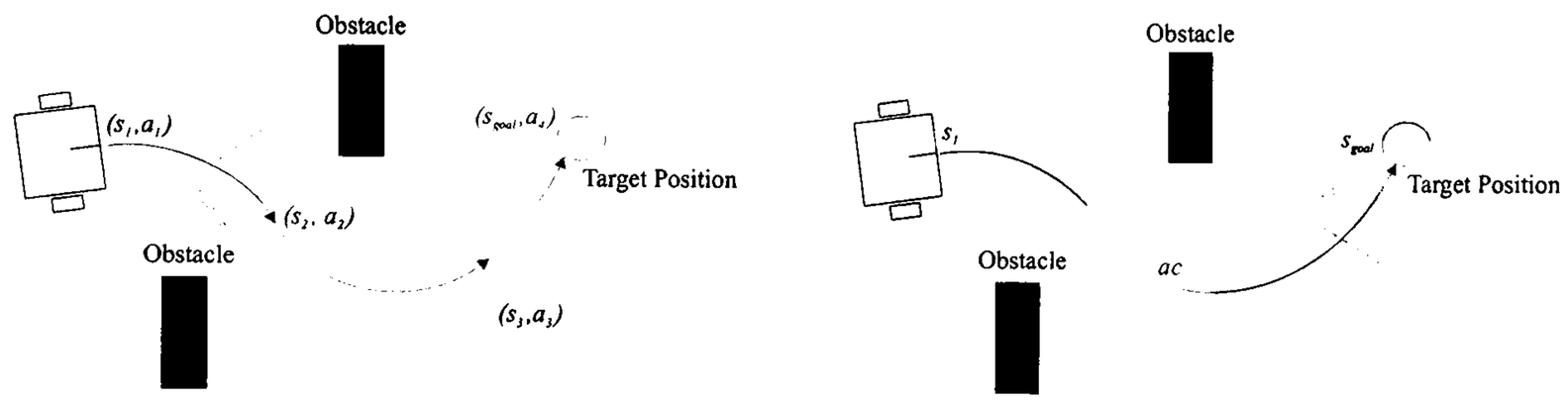
(b) State concept representation of navigation history

Figure 7-2: State concept representation of navigation history

Any of the previous state concepts is related to an ordered sequence of actions, or an action concept, generated using the temporal relations. As seen in Section 4.4.1 these can be regarded as *plans*.

Assuming that a robot's current state and its goal state are contained in a previously observed state concept, then the action concept related to the state concept could be used to control the robot. For example, Figure 7.3(a) illustrates a robot's trajectory from an initial position to a target position

when controlled using atomic states and atomic actions. The initial state in the figure is s_1, and the goal state is s_{goal}. Given this example, the robot could have acquired the following relational concepts: $sc = \langle s_1, s_2, s_3, s_{goal}; R_t \rangle$ and $ac = \langle a_1, a_2, a_3, a_4; R_t \rangle$, where R_t indicates that the relation between primitives is temporal. Figure 7.3(b) illustrates a new situation, in which the robot's current state is s_1, and its goal state is s_{goal}. The previously observed sc, is a state concept that includes the initial and goal states for the new situation. Thus, action concept ac could be used as a control action.



(a) Atomic action control

(b) Action concept control

Figure 7-3: Robot controlled at different description levels

In Figure 7.3(a) the control strategy is similar to the control exerted by reactive architectures (see Section 2.2.2), in which a robot reacts to the perceived state with an action dictated by a behaviour. Instead, controlling a robot using action concepts as illustrated in Figure 7.3(b), is equivalent to navigating with extended actions, or using plans. For example, selecting an action concept defined by three atomic actions can be seen as using a three-step plan. As reviewed in Section 2.2.1, deliberative architectures operate by creating and executing plans, although the mechanism used by deliberative architectures to construct plans is not based on hierarchical action and state aggregation, but on logic and planning techniques.

The notion of temporal concepts may produce a way to integrate planning and reactive control strategies for robotics. Moreover, as action concepts or *plans* can be learned from experience, this approach may allow robots to be designed with simple reactive behaviours and with capabilities of acquiring planning dynamically. An obvious difficulty in implementing this approach would be that of *closed loop control*, as long action concepts would mean long periods of action without feedback from the environment. To address this problem, the atomic states within the state concept could be used as sub-goals.

7.3.2 Extension to behaviour learning with concepts

In this thesis, behaviours were learned using supervised methods, that is, by providing the robot with examples of the desired behaviour. Although this approach was used to demonstrate that behaviour learning can be undertaken using concepts in a more general context, robots should learn using only their own experience. That is, robots should use a framework, such as RL, in which a robot learns solely by using its interaction with the environment. This approach would result in an architecture similar to the one presented in Chapter 4, with the difference that the behaviour learning and control component would be replaced by a reinforcement learning component, as illustrated in Figure 7-4. As previously in this architecture, the concept generation component generates state and action concepts using different classification techniques, such as cluster-based or relational-based. These concepts are then used by the reinforcement learning component to learn control policies. As in the RL approach, the robot would be controlled

simultaneously while learning the control policy, thus resulting on an autonomous, unsupervised learner.

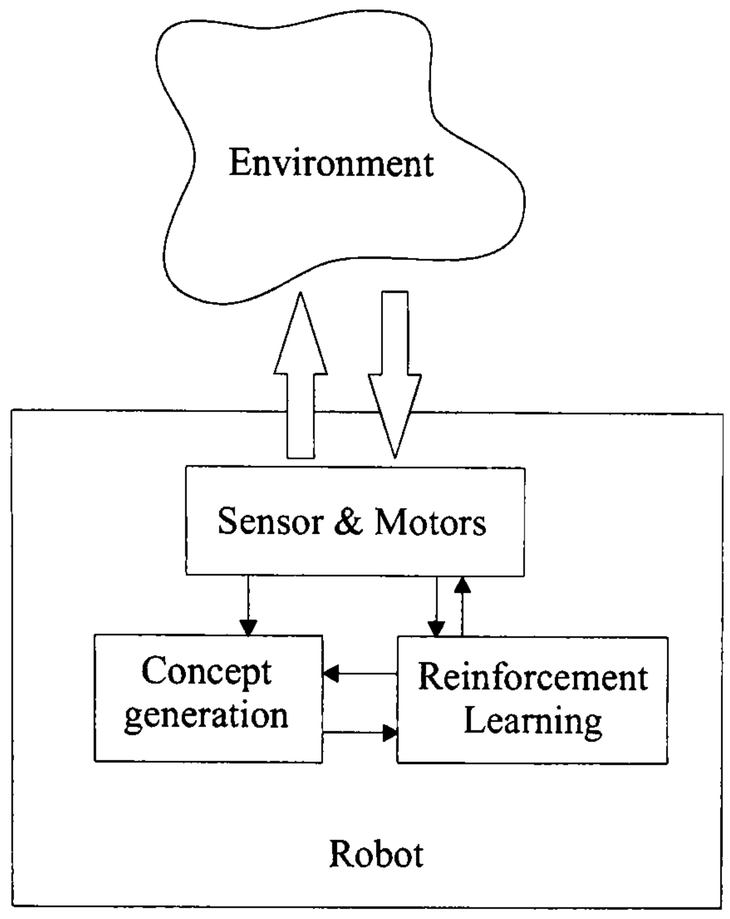


Figure 7-4: Extension to the architecture

A direct implication of this approach, is that concepts can be generated on the fly while learning policies. Thus, the methods used to generate concepts should be adaptive (similarly to the one presented in Section 5.6). With the integration of RL, generalisation concepts could be defined using splitting criteria similar to the ones used in variable resolution approaches (see Section 2.4.2).

Moreover, the introduction of temporally-related concepts, as described in the previous section, would allow the RL methods to exploit *time-extended* actions. That is, frameworks such as *options* [Sutton *et al.*, 1999, Precup, 2000] could be implemented for behaviour learning. In the options framework, temporally-extended actions or options are defined using three elements, (I, π, β), where I is an initialisation state, π is the policy of the time-extended

action, and β is a termination condition. If I is satisfied then the option can be started. During its execution, the option follows policy π, and the execution is stopped when β is satisfied. In our concept-based approach, the initialisation state I, would be equivalent to the initial atomic state within a state concept. The policy π, would be the action concept which indicates the atomic actions to select. The termination condition β, would be associated with the last atomic state in the state concept.

7.3.3 Emergence and evolution of grounded communication

Unlike the research into how language and communication could emerge and evolve within a group of autonomous robots is an interesting and new research field [Steels, 2003, Steels and Baillie, 2003]. This thesis defined concepts strictly as abstract representations used for behaviour learning and robot control, ignoring their *semantics* and *ontological* characteristics. That is, the meaning of concepts and their relations were not addressed.

In the research presented in [Steels, 2003, Steels and Baillie, 2003], groups of robots are situated in an environment and learn a communication protocol, i.e. words and meanings (semantics), by interacting with the environment and with the other robots in the group. That is, the meaning of words are grounded based on the robot's sensory-motor data, the robot-environment interaction and the robot-robot interaction. This way of defining words and their meaning is very similar to the way concepts are generated in this thesis. Thus, a possible extension of the work presented in this thesis would be that of using concepts as communication tools in groups of robotic agents.

In the research related to the emergence and evolution of communication. robots interact with the environment and with each other by *playing language games* [Steels, 1998, Vogt, 2001]. In general, these games are based on having a robot describing an object and another robot trying to guess what is the object being described. These games require two processes: (i) object discrimination and description (discrimination games [Steels, 1996]), and (ii) description communication. The first process must be capable of finding the features that 'best' describe the object in focus. The second process must be capable of transmitting and interpreting features.

The first process is that of finding relevant features related to a particular class. As shown in this thesis, the methods based on Q-analysis can yield new insights into the issue of feature or variable selection. Thus, it would be interesting to use those methods in an application such as that of emergence and evolution of communication.

Appendix A

Glossary

This thesis uses a number of technical terms which are described in the following glossary.

Action: is a command that an agent can execute, usually related to its motor capabilities. An action executed at time t is denoted by a_t. A set of actions is denoted by A.

Atomic Action: if actions are described at various description levels, then atomic actions are the ones at the lowest level of description. In other words, atomic actions can not be decomposed into simpler actions. For example, the power sent to the motors of a robot constitutes an atomic action.

Atomic State: if states are represented at various levels of description, then atomic states are the ones at the lowest level of description. In other words, atomic states can not be decomposed into simpler states. For example the readings from a robot's sensors describe atomic states.

Behaviour: is the observable result of an agent acting in its environment. This is usually the emergent result of the interaction of the behaviour function, the robot and the environment.

Behaviour function: is an agent's internal function that maps the perceived environmental states onto actions, $B : S \rightarrow A$.

Concept: a general class formed of primitives.

Primitive: in the context of classification, the particular elements of the set to be classified are called primitives.

Reward: used in reinforcement learning to indicate the goodness of the system's state transition. For example, a mobile robot colliding with an obstacle could receive a negative reward as this is an undesirable state transition (from not colliding to colliding). The reward received at time t is denoted by r_t, and is also known as immediate reward.

State: is a description of the environment as perceived by the agent through its sensors (sonar, cameras, encoders, etc). A state observed at time t is denoted by s_t. For example, $s_t = (coordinate\_position, relative\_speed, number\_obstacles)$ could represent the state of the environment observed by a mobile robot. Each of the variables used to define a state, e.g. *coordinate_position*, *relative_speed*, and *number_obstacles* are known as *state variables*. A set of states is denoted by S.

State transition: given that s_t is the state observed at time t and s_{t+1} is the one observed at time $t + 1$, the transition from s_t to s_{t+1} is a state transition.

References

[Albus, 1975] J. S. Albus. Data storage in the cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement and Control*, 97(3):228 – 233, 1975.

[Andre and Russell, 2002] D. Andre and S. J. Russell. State abstraction for programmable reinforcement learning agents. In *Eighteenth National Conference on Artificial Intelligence (AAAI- 02)*, pages 119–125, Cambridge, MA, 2002. The MIT Press.

[Arkin, 1987] R. Arkin. Motor schema based navigation for a mobile robot: An approach to programming by behavior. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 264–271, 1987.

[Arkin, 1989] R. C. Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotic Research*, 8:92–96, 1989.

[Arkin, 1998] R. C. Arkin. *Behavior-Based Robotics.* The MIT Press, Cambridge, MA, 1998.

[Atkeson *et al.*, 1997a] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997.

[Atkeson *et al.*, 1997b] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 11:75–113. 1997.

[Atkin, 1977] R. H. Atkin. *Combinatorial connectivities in social systems.* Birkhäuser Verlag, Basel, 1977.

[Atkin, 1981] R. H. Atkin. *Multidimensional Man.* Penguin Books, Harmondsworth, 1981.

[Barto and Mahadevan, 2003] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discreet Event Dynamic Systems: Theory and Applications*, 13:41–77, 2003.

[Barto *et al.*, 1983] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):835–846, 1983.

[Bellman, 1957] R. E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[Benson and Nilsson, 1995] S. Benson and N. J. Nilsson. Reacting, planning and learning in an autonomous agent. In K. Furukawa, S. Muggleton, and D. Michie, editors, *Machine Intelligence*, volume 14, pages 29–64. Oxford University Press, Oxford, UK, 1995.

[Bishop, 1995] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Oxford, UK, 1995.

[Bonasso *et al.*, 1997] R. P. Bonasso, R. J. Firby, E. Gat, D.K.D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive

systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2-3):237–256, 1997.

[Braitenberg, 1986] V. Braitenberg. *Vehicles, Experiments in Synthetic Psychology.* The MIT Press, Cambridge, MA, 1986.

[Bratman *et al.*, 1988] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4):349–355, 1988.

[Brooks and Mataric, 1993] R. A. Brooks and M. J. Mataric. Real robots, real learning problems. In J. H. Connell and Sridhar Mahadevab, editors, *Robot Learning*, pages 193–213. Kluwer Academic Publishers, Boston, MA, 1993.

[Brooks, 1985] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1985.

[Brooks, 1990] R. A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6(1&2):3–15, 1990.

[Brooks, 1991] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[Cao *et al.*, 1997] U. Y. Cao, A. S. Fukunaga, and A. B. Kahng. Cooperative mobile robotics: antecedents and directions. *Autonomous Robots*, 4(1):7–27, 1997.

[Carbonell and Gil, 1990] J. G. Carbonell and Y. Gil. Learning by experimentation: the operator refinement method. In Y. Kodratoff and R.S.

Michalski, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 3. Morgan Kaufmann Publishers, San Mateo, California, 1990.

[Carbonell, 1983] J. G. Carbonell. Learning by analogy: Formulating and generalizing plans from past experience. In R. S. Michalsky, T. M. Mitchell, and J. G. Carbonell, editors, *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann Publishers, San Mateo, California, 1983.

[Chandrasekaran and Goel, 1988] B. Chandrasekaran and A. Goel. From numbers to symbols to knowledge structures: Artificial intelligence perspectives on the classification task. *IEEE Tansactions on Systems, Man and Cybernetics*, 18(3):415–424, 1988.

[Chapman and Kaelbling, 1991] D. Chapman and L. P. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *12th International Joint Conference on Artificial Intelligence*, pages 726–731, Sydney, Australia, 1991.

[Chella *et al.*, 1997a] A. Chella, M. Frixione, and S. Gaglio. A cognitive architecture for artificial vision. *Artificial Intelligence*, 89(1&2):73–111, 1997.

[Chella *et al.*, 1997b] A. Chella, S. Gaglio, G. Sajeva, and F. Torterolo. An architecture for autonomous agents integrating symbolic and behavioural processing. In *IEEE EUROMICRON Workshop on Advanced Mobile Robots*, pages 45–50, 1997.

[Chella *et al.*, 1998] A. Chella, M. Frixione, and S. Gaglio. An architecture

for autonomous agents exploiting conceptual representations. *Robotics and Autonomous Systems*, 25(3):231–240, 1998.

[Chen *et al.*, 2002] M. Chen, E. Foroughi, F. Heintz, Z. Huang, S. Kapetanakis, K. Kostiadis, J. Kummeneje, I. Noda, O. Obst, P. Riley, T. Steffens, Y. Wang, and X. Yin. *RoboCup Soccer Server Manual*, 2002.

[Chow and Tsitsiklis, 1991] C. S. Chow and J.N. Tsitsiklis. An optimal one-way multigrid algorithm for discrete-time stochastic control. *IEEE Transactions on Automatic Control*, 36(8):898–914, 1991.

[Chrisley and Ziemke, 2002] R. Chrisley and T. Ziemke. Embodiment. In *Encyclopedia of Cognitive Science*. Macmillan, London, 2002.

[Connell, 1992] J. H. Connell. SSS: A hybrid architecture applied to robot navigation. In *IEEE International Conference on Robotics and Automation*, pages 2719–2724, 1992.

[Coste-Manière and Simmons, 2000] È. Coste-Manière and R. Simmons. Architecture, the backbone of robotic systems. In *IEEE International Conference on Robotics and Automation*, pages 67–72, San Francisco, CA, 2000.

[Darken and Moody, 1990] C. Darken and J. Moody. Fast, adaptive k-means clustering: Some empirical results. In *IEEE International Joint Conference on Neural Networks*, pages 233 – 238, San Diego, CA, 1990.

[Dash and Liu, 1997] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(3):131–156, 1997.

[Dietterich, 1998] T. G. Dietterich. The MAXQ method for hierarchical reinforcement learning. In *15th International Conf. on Machine Learning*, pages 118–126, San Francisco, CA, 1998. Morgan Kaufmann Publishers Inc.

[Duda and Hart, 1973] R. O. Duda and P. E. Hart. *Pattern classification and sceen analysis.* John Wiley & Sons, New York, 1973.

[Feinstein, 1996] A. R. Feinstein. *Multivariable Analysis an Introduction.* Yale University Press, New Haven, CT, 1996.

[Ferguson, 1992] I. A. Ferguson. *TouringMachines An Architecture for Dynamic, Rational, Mobile Agents.* PhD thesis, University of Cambridge, Cambridge, UK, 1992.

[Fikes and Nilson, 1971] R.E. Fikes and N. Nilson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3&4):189–208, 1971.

[Fisher, 1936] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

[Fisher, 1987] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2(2):139–172, 1987.

[Gat, 1992] E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *AAAI Proceedings of the National Conference on Artificial Intelligence*, pages 809–815, San Jose, California, 1992.

[Gat, 1998] E. Gat. Three-layer architectures. In *Artificial intelligence and mobile robots: case studies of successful robot systems*, pages 195–210. MIT Press, Cambridge, MA, USA, 1998.

[Gennari *et al.*, 1989] J. H. Gennari, P. Langley, and D. Fisher. Models of incremental concept formation. *Artificial Intelligence*, 40(1-3):11–61, 1989.

[Georgeff and Lansky, 1987] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *6th National Conference on Artificial Intelligence (AAAI)*, pages 677–682, Seattle, WA, 1987.

[Georgeff, 1987] M. P. Georgeff. Planning. *Annual Review of Computer Science*, 2:359–400, 1987.

[Goldberg and Mataric, 1999] D. Goldberg and M. J. Mataric. Coordinating mobile robot group behavior using a model of interaction dynamics. In *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 334–349, Seattle, WA, 1999.

[Gordon, 1999] A. D. Gordon. *Classification.* Monographs on Applied Probability and Statistics. Chapman and Hall, Boca Raton, Florida, 2nd edition, 1999.

[Harnad, 1990] S. Harnad. The symbol grounding problem. *Physica D*, 42:335–346, 1990.

[Hesselroth *et al.*, 1994] T. Hesselroth, K. Sarkar, P. Van der Smagt, and Schulten K. Neural network control of a pneumatic robot arm. *IEEE Transactions on Systems, Man and Cybernetics*, 24(1):28–38, 1994.

[Hu and Brady, 1995] H. Hu and M. Brady. Application of parallel processing to intelligent control of mobile robots. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2910–2915, 1995.

[Hu and Brady, 1996] H. Hu and M. Brady. A parallel processing architecture for sensor-based control of intelligent mobile robots. *Robotics and Autonomous Systems*, 17(4):235–257, 1996.

[Hu and Gu, 2005] H. Hu and D. Gu. Hybrid learning architecture for fuzzy control of quadruped walking robots. *International Journal of Intelligent Systems*, 20(2):131–152, 2005.

[Ingrand *et al.*, 1992] F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6):34–44, 1992.

[Iravani *et al.*, 2004] P. Iravani, L. Rapanotti, and J. H. Johnson. Application of concept grounding techniques to reduce dimensionality in sensory-motor spaces. In E. Onaindia and S. Staab, editors, *Second European Starting AI Researchers' Symposium STAIRS*, Valencia, Spain, 2004. IOS press.

[Iravani *et al.*, 2005] P. Iravani, J. H. Johnson, and L. Rapanotti. Robotics and the Q-analysis of behaviour. In *Tenth International Symposium on Artificial Life and Robotics AROB*, Beppu, Japan, 2005.

[Iravani, 2004] P. Iravani. Behaviour-based architecture for abstract learning

and control. In *Towards Autonomous Robotic Systems TAROS*, Colchester, UK, 2004.

[Iravani, 2006] P. Iravani. Detecting irrelevant sensor data by Q-analysis. In *RoboCup-05: Robot Soccer World Cup IX*, Osaka, Japan, 2006. Springer.

[Jain *et al.*, 1999] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[Jain *et al.*, 2000] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical pattern recognition: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

[Jennings *et al.*, 1998] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1:7–38, 1998.

[John *et al.*, 1994] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *11th International Conference on Machine Learning*, pages 121–129, New Brunswick, NJ, 1994.

[Johnson and Picton, 1990] J. H. Johnson and P. Picton. *Mechatronics: Designing Intelligent Machines*, volume 2. Butterworth-Heinemann and The Open University, 1990.

[Johnson, 1981] J. H. Johnson. Some structures and notation of Q-analysis. *Environment and Planning*, 8:73–86, 1981.

[Johnson, 1983] J. H. Johnson. Hierarchical set definition by Q-analysis, part I. the hierarchical backcloth. *International Journal of Man-Machine Studies*, 18:337–359, 1983.

[Johnson, 1986] J. H. Johnson. Stars, maximal rectangles and lattices: a new perspective on Q-analysis. *International Journal of Man-Machine Studies*, 24:293–299, 1986.

[Kaelbling and Rosenschein, 1990] L. P. Kaelbling and S. J. Rosenschein. Action and planning in embedded agents. In Pattie Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 35–48. The MIT Press, Cambridge, MA, 1990.

[Kaelbling *et al.*, 1996] L. P. Kaelbling, M. L. Littman, and A. W. More. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[Kendall, 1980] M. Kendall. *Multivariate Analysis.* Griffin, London, UK, 2nd edition, 1980.

[Kohonen, 1995] T. Kohonen. *Self-Organizing Maps.* Springer-Verlag, Berlin, 3rd edition, 1995.

[Kostiadis and Hu, 2001] K. Kostiadis and H. Hu. Kabage-rl: Kanerva based generalisation and reinforcement learning for possession football. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 292–297, Hawaii, USA, 2001.

[Kumar and Fogel, 1999] C. Kumar and D. B. Fogel. Evolving neural networks to play checkers without relying on expert knowledge. *IEEE Transactions on Neural Networks*, 10(6):1382–1391, 1999.

[Kuvayev and Sutton, 1997] L. Kuvayev and R. S. Sutton. Approximation

in model-based learning. In *ICML97 Workshop on Modelling in Reinforcement Learning*, Nashville, TN, 1997.

[Lebowitz, 1987] M. Lebowitz. Experiments with incremental concept formation: Unimem. *Machine Learning*, 2(2):103–138, 1987.

[Maes and Brooks, 1990] P. Maes and R. A. Brooks. Learning to coordinate behaviors. In *National Conference on Artificial Intelligence AAAI*, pages 796–802, Boston, MA, 1990.

[Maes, 1989] P. Maes. The dynamics of action selection. In *International Joint Conference on Artificial Intelligence*, pages 991–997, Detroit, Michigan, 1989.

[Maes, 1990] P. Maes. Situated agents can have goals. In P. Maes, editor, *Designing Autonomous Agents*, pages 49–70. The MIT Press, Cambridge, MA, 1990.

[Mahadevan and Conell, 1991] S. Mahadevan and J. H. Conell. Automatic programming of behaviour-based robots using reinforcement learning. In *Ninth National Conference on Artificial Intelligence AAAI*, Anaheim, California, 1991.

[Mataric, 1992] M. J. Mataric. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312, 1992.

[Mataric, 1997] M. J. Mataric. Behavior-based control: Examples from navigation, learning, and group behaviour. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2-3):323–336, 1997.

[Mataric, 1999] M. J. Mataric. Behavior-based robotics. In Robert A. Wilson Keil and Frank C., editors, *MIT Encyclopedia of Cognitive Sciences*, pages 74–77. The MIT Press, cambridge, MA, 1999.

[Mataric, 2001] M. J. Mataric. Learning in behavior-based multi-robot systems: policies, models, and other agents. *Cognitive Systems Research*, 2(1):81–93, 2001.

[Merriam-Webster, 2004] Merriam-Webster. Merriam-webster online. http://www.m-w.com/, 2004.

[Michalski and Stepp, 1983] R. S. Michalski and R. Stepp. Learning from observation: Conceptual clustering. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume II, pages 331–363. TIOGA Publishing Co., Palo Alto, 1983.

[Michalski, 1980] R. S. Michalski. Knowledge acquisition through conceptual clustering: A theoretical framework and an algorithm for partitioning data into conjunctive concepts. *International Journal of Policy Analysis and Information Systems*, 4(3):219–243, 1980.

[Michaud and Mataric, 1999] F. Michaud and M. J. Mataric. Representation of behavioral history for learning in nonstationary conditions. *Robotics and Autonomous Systems*, 29(2):187–200, 1999.

[Michell, 1997] T. M. Michell. *Machine Learning.* McGraw-Hill, New York, 1997.

[Moore and Atkeson, 1993] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130, 1993.

[Moore and Atkeson, 1995] A. W. Moore and C. G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21(3):199–233, 1995.

[Moore, 1991] A. W. Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. In L. Birnbaum and G. Collins, editors, *8th International Conference on Machine Learning*, San Francisco, CA, 1991.

[Munos and Moore, 1999] R. Munos and A. W. Moore. Variable resolution discretization for high-accuracy solutions of optimal control problems. In *16th International Joint Conference on Artificial Intelligence*, pages 1348–1355, Stockholm, Sweden, 1999.

[Nehmzow, 2003] U. Nehmzow. *Mobile Robotics: A Practical Introduction.* Springer, Berlin, second edition, 2003.

[Newell and Simon, 1976] A. Newell and H. A. Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19(3):113–125, 1976.

[Nicolescu and Mataric, 2002] M. N. Nicolescu and M. J. Mataric. A hierarchical architecture for behavior-based robots. In *Autonomous Agents and Multi Agent Systems*, pages 227–233, Bologna, Italy, 2002.

[Noda *et al.*, 1998] I. Noda, H. Matsubara, K. Hiraki, and I. Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12(2):233–250, 1998.

[Peng and Williams, 1993] J. Peng and R. J. Williams. Efficient learning and planning within the Dyna framework. *Adaptive Behavior*, 1(4):437–454, 1993.

[Pfeifer and Scheier, 1997] R. Pfeifer and C. Scheier. Sensory-motor coordination: the metaphor and beyond. *Robotics and Autonomous Systems*, 20(2):157–178, 1997.

[Pfeifer and Scheier, 2001] R. Pfeifer and C. Scheier. *Understanding Intelligence*. The MIT Press, Cambridge, MA, 2001.

[Precup, 2000] D. Precup. *Temporal Abstraction in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst, Amherst, 2000.

[Rendell, 1986] L. Rendell. A general framework for induction and a study of selective induction. *Machine Learning*, 1(2):177–226, 1986.

[Reynolds, 2000] S. I. Reynolds. Adaptive resolution model-free reinforcement learning: Decision boundary partitioning. In *7th International Conference on Machine Learning*, pages 783–790, Stanford, CA, 2000.

[Ribeiro, 2002] C. Ribeiro. Reinforcement learning agents. *Artificial Intelligence Review*, 17(3):223–250, 2002.

[Rosenschein and Kaelbling, 1986] S. J. Rosenschein and L. P. Kaelbling. The synthesis of digital machines with provable epistemic properties. In

Conference on Theoretical Aspects of Reasoning About Knowledge, pages 83–98, Monterey, CA, 1986.

[Rosenschein and Kaelbling, 1995] S. J. Rosenschein and L. P. Kaelbling. A situated view of representation and control. *Artificial Intelligence*, 73(1-2):149–173, 1995.

[Santamaría *et al.*, 1998] J. C. Santamaría, R. S. Sutton, and A. Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behaviour*, 6(2):163–217, 1998.

[Schaal and Atkeson, 1994] S. Schaal and C. G Atkeson. Robot juggling: Implementation of memory-based learning. *Control Systems Magazine*, 14(1):57–71, 1994.

[Simons *et al.*, 1982] S. Simons, H. Van Brussel, J. De Schutter, and J. Verhaert. A self-learning automaton with variable resolution for high precision assembly by industrial robots. *IEEE Transactions on Automatic Control*, 27(5):1109–1113, 1982.

[Smart and Kaelbling, 2000] W. D. Smart and L. P. Kaelbling. Practical reinforcement learning in continuous spaces. In *7th International Conference on Machine Learning*, pages 903–910, San Francisco, CA, 2000.

[Smart and Kaelbling, 2002] W. D. Smart and L. P. Kaelbling. Effective reinforcement learning for mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 3404–3410, Washington, DC, 2002.

[Steels and Baillie, 2003] L. Steels and J. Baillie. Shared grounding of event

descriptions by autonomous robots. *Robotics and Autonomous Systems*, 43(2-3):163–173, 2003.

[Steels, 1995a] L. Steels. Building agents out of autonomous behavior systems. In Luc Steels and Rodney Brooks, editors, *The Artificial Life Route to Artificial Intelligence: Building Embodied, Situated Agents*, pages 83–121. Lawrence Erlbaum, Hillsdale, New Jersey, 1995.

[Steels, 1995b] L. Steels. When are robots intelligent autonomous agents? *Journal of Robotics and Autonomous Systems*, 15(1):3–9, 1995.

[Steels, 1995c] R. A. Steels, L.and Brooks, editor. *The Artificial Life Route to Artificial Intelligence: Building Embodied, Situated Agents.* Lawrence Erlbaum, Hillsdale, New Jersey, 1995.

[Steels, 1996] L. Steels. Perceptually grounded meaning creation. In *International Conference of Multi-Agent Systems, ICMAS96*, Kyoto, Japan, 1996.

[Steels, 1998] L. Steels. The origins of syntax in visually grounded robotic agents. *Artificial Intelligence*, 103(1-2):133–156, 1998.

[Steels, 2003] L. Steels. Evolving grounded communication for robots. *Trends in Cognitive Sciences*, 7(7):308–312, 2003.

[Stefan *et al.*, 2000] S. Stefan, C. G. Atkeson, and S. Vijayakumar. Real-time robot learning with locally weighted statistical learning. In *IEEE International Conference on Robotics and Automation*, pages 288–293. San Francisco, California, 2000.

[Stone and Sutton, 2001] P. Stone and R. S. Sutton. Scaling reinforcement learning toward robocup soccer. In *8th International Conference on Machine Learning*, pages 537–544, San Francisco, CA, 2001.

[Stone, 1998] P. Stone. *Layered Learning in Multi-Agent Systems.* PhD thesis, Carnegie Mellon University, 1998.

[Sutton and Barto, 1998] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, Cambridge, MA, 1998.

[Sutton *et al.*, 1999] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[Sutton, 1988] R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[Sutton, 1990] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *7th International Conference on Machine Learning*, pages 216–224, Stanford University, 1990.

[Sutton, 1991] R. S. Sutton. Dyna, an integrated architecture for learning, planning and reacting. In *Working Notes of the 1991 AAAI Spring Symposium*, pages 151–155, 1991.

[Sutton, 1996] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems*, 8:1038–1044, 1996.

[Tsitsiklis and Van Roy, 1996] J. N. Tsitsiklis and B. Van Roy. Feature-based methods for large scale dynamic programming. *Machine Learnig*, 22(1-3):59–94, 1996.

[Vesanto and Alhoniemi, 2000] J. Vesanto and E. Alhoniemi. Clustering of the self-organizing map. *IEEE Transactions on Neural Networks*, 11(3):586–600, 2000.

[Vogt, 2001] P. Vogt. Bootstrapping grounded symbols by minimal autonomous robots. *Evolution of Communication*, 4(1):87–116, 2001.

[Vollbrecht, 1999] H. Vollbrecht. KD-Q-learning with hierarchic generalization in action- and state-space. In *Web-Veröffentlichung zum European Workshop of Reinforcement Learning EWRL-4*, Lugano, 1999.

[Wactkins and Dayan, 1992] C.J.C.H Wactkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[Wactkins, 1989] C.J.C.H Wactkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, UK, 1989.

[Walter, 1953] G. W. Walter. *The Living Brain.* Pinguin, London, 1953.

[Weiss, 1999] G. Weiss, editor. *Multiagent Systems a Modern Approach to Distributed Artificial Intelligence.* The MIT Press, Cambridge, MA, 1999.

[Wooldridge and Jennings, 1995] M. J. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practise. *Knowledge Engineering Review*, 10(2):115–152, 1995.

[Wu and Chow, 2004] S. Wu and T.W.S. Chow. Clustering of the self-organizing map using a clustering validity index based on inter-cluster and intra-cluster density. *Pattern Recognition*, 37:175–188, 2004.

Index